UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS

MULTI-SKILL RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEMS: MODELS AND ALGORITHMS

Doutoramento em Estatística e Investigação Operacional
Especialidade de Otimização

Bernardo Pedro Esteves Ferreira de Almeida

Tese orientada por:
Prof. Doutor Francisco Alexandre Saldanha da Gama Nunes da Conceição
Prof. Doutora Isabel Cristina Silva Correia

Documento especialmente elaborado para a obtenção do grau de doutor

2017

UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

MULTI-SKILL RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEMS: MODELS AND ALGORITHMS

Doutoramento em Estatística e Investigação Operacional
Especialidade de Otimização

Bernardo Pedro Esteves Ferreira de Almeida

Tese orientada por:
Prof. Doutor Francisco Alexandre Saldanha da Gama Nunes da Conceição
Prof. Doutora Isabel Cristina Silva Correia

Júri:
Presidente:

- Doutor Luís Eduardo Neves Gouveia, Professor Catedrático
  Faculdade de Ciências da Universidade de Lisboa

Vogais:

- Doutor Jorge Manuel Pinho de Sousa, Professor Catedrático
  Faculdade de Engenharia da Universidade do Porto
- Doutor António José Galrão Ramos, Professor Adjunto
  Instituto Superior de Engenharia do Porto, do Instituto Politécnico do Porto
- Doutor Luís Eduardo Neves Gouveia, Professor Catedrático
  Faculdade de Ciências da Universidade de Lisboa
- Doutora Maria Eugénia Vasconcelos Captivo, Professora Catedrática
  Faculdade de Ciências da Universidade de Lisboa
- Doutor Francisco Alexandre Saldanha da Gama Nunes da Conceição, Professor Auxiliar
  Faculdade de Ciências da Universidade de Lisboa

Documento especialmente elaborado para a obtenção do grau de doutor

2017

Abstract

In this dissertation, project scheduling problems with multi-skill resources are investigated. These problems are commonly found in companies making use of human resources or multi-purpose machinery equipment. The general problem consists of a single project comprising a set of activities. There are precedence relations between the activities. Each activity requires one or several skills for being processed and for each of these skills, more than one resource may be needed. The resources have a unitary capacity per time unit and may master more than one skill. The resources can contribute with at most one skill to at most one activity that requires it, in each time unit. It is usually assumed that the resources are homogeneous, i.e., the proficiency at which each skill is performed is the same across all resources that master that skill. Preemption is not allowed, which implies that once an activity starts being processed it cannot be interrupted. When a resource is assigned to perform a skill for an activity, it remains in that status for the whole processing time of the activity. The objective of the problem is to schedule all the activities, satisfying all constraints such that the makespan of the project is minimized.

After introducing a framework to the realm of project scheduling problems with multi-skill resources and highlighting the main objectives and contributes of this thesis, a state-of-the-art review on the topic is presented.

The particular problem investigated in this document is then described in detail and its specific features are discussed. To that end, a continuous-time mathematical formulation from the literature is revisited, an example of the problem is presented and some aspects related to the computation of feasible solutions are discussed. This last topic is of major relevance when dealing with problems that combine personnel staffing with project scheduling.

In order to properly assess the quality of solutions obtained by the methodological developments proposed in this thesis, it became necessary to develop an instance generator to build a set of instances larger than those existing in the literature. After formally proposing such generator, we detail the characteristics of the two sets of instances considered for the computational experiments to be performed.

In the next sections of the document, the solution methodologies developed within the scope of this thesis are presented and thoroughly discussed.

A wide range of mathematical formulations is studied, two of which are first proposed in this document. From the assessment of their ability both to compute feasible and possibly optimal solutions and to derive good lower bounds (stemming from their linear programming relaxations) to the problem, it will become clear that the so-called discrete-time formulations yield the strongest lower bounds whereas a continuous-time formulation from the literature proved to be the most suitable for solving instances of the problem to optimality. This trend is observed for both sets of instances considered. Two constructive

lower bound mechanisms proposed for the resource-constrained project scheduling problem are extended to account for the existence of multi-skill resources and multi-skill requirements of the activities. The results reveal that such methods improve the lower bounds achieved by the studied mathematical formulations for some instances.

Real-world project scheduling problems usually involve a large number of activities, resources and skills. Hence, the use of exact methods such as the standard techniques for tackling the aforementioned mathematical models, is often impractical. When faced with this kind of situations, a project manager may consider preferable to have a good feasible solution, not necessarily an optimal one, within an admissible time, by means of an approximate method.

A close look into the problem being investigated in this thesis reveals that it has some features that are not present in some well-studied particular cases of it, namely the notion of skill—multi-skill resources and skill requirements of the activities. Hence, with the objective of developing approximate solution methodologies that better exploit the specific characteristics of the problem at hand, two new concepts are introduced: activity grouping and resource weight. The well-known parallel and serial scheduling schemes, proposed originally for the class of resource-constrained project scheduling problems, are extended to our problem setting and the two above-mentioned concepts are incorporated into these two new frameworks. Such frameworks use well-known activity priority rules for defining the order by which the activities are selected to be scheduled and resource weight rules to determine a set of resources that meets the requirements of all the activities to be scheduled at each time with the least total cost (weight). Thereafter, two heuristic procedures making use of those schedule generation schemes are proposed, namely a multi-pass heuristic built upon the parallel scheduling scheme and a biased random-key genetic algorithm. The idea of computing a feasible solution using the so-called backward planning is also considered in both methods.

The multi-pass heuristic retrieves the solution with the minimum makespan after performing a specific number of passes, each associated with a unique combination of the considered activity priority rules, the developed resource weight rules and the two precedence networks: forward and backward.

The biased random-key genetic algorithm is a metaheuristic whose developed chromosome structure encodes information related to: (i) the priority values of the activities; (ii) the weights of the resources; (iii) how a chromosome is decoded, i.e., the scheduling scheme and precedence network scheme to be used for computing the associated makespan. By embedding all this information into the chromosomes, it becomes possible to take advantage of the evolutionary framework of the biased random-key genetic algorithm, which tends to allow the evolution of such data (change in their values) over time, towards better makespan valued solutions. Three variants of the biased random-key genetic algorithm are considered with regard to the type of scheduling generation scheme to be used for decoding its chromosomes: (i) all chromosomes are decoded with the parallel scheduling scheme; (ii) all chromosomes are decoded with the serial scheduling scheme; (iii) the scheduling scheme to be used for decoding each chromosome depends on the value of the associated parameter which is embedded in the chromosome.

The computational results revealed that the proposed multi-pass heuristic is an efficient algorithm for computing feasible solutions of acceptable quality within a small computational time whereas the biased random-key genetic algorithm is a robust algorithm and a more competitive approximate approach for computing feasible solutions of higher quality, especially for harder instances such as those of medium and large dimensions.

We conclude this thesis with an overview of the work done and with some directions for further research in terms of methodological developments and of some potentially interesting extensions of the addressed problem.

Keywords: Project scheduling; multi-skill resources; optimization models; heuristics; lower bounds.

Resumo

Nesta dissertação investigam-se problemas de sequenciamento de projetos com recursos limitados e polivalentes (*multi-skill*), i.e., recursos capazes de desempenhar diversas funções, estando cada uma delas associada à realização de uma competência (*skill*) específica. Estes problemas são habitualmente encontrados em empresas que trabalhem com recursos humanos ou máquinas cuja utilização possa satisfazer várias finalidades distintas. Tipicamente, um problema deste tipo consiste num projeto que é composto por um conjunto de atividades, com determinadas durações e entre as quais existem relações de precedência. Cada atividade necessita de uma ou mais competências para ser processada e para cada uma dessas competências poderá ser preciso recorrer à alocação de mais do que um recurso. Os recursos têm uma capacidade unitária em cada unidade de tempo e detêm, normalmente, mais do que uma competência. Assume-se ainda que os recursos são homogéneos, i.e., a eficiência com que cada competência é realizada é a mesma para todos os recursos que a saibam desempenhar. Não é permitida a suspensão da execução das atividades, o que implica que uma vez que uma atividade comece a ser executada, não possa ser interrompida. Quando um recurso é afeto à realização de uma determinada competência para uma atividade, ele permanece a executar essa mesma competência durante todo o tempo de processamento da atividade. O problema tem como objetivo o sequenciamento de todas as atividades, satisfazendo todas as restrições, de forma a que o projeto termine o mais cedo possível.

Após realizar uma breve introdução destinada ao enquadramento da área de investigação de problemas de sequenciamento de projetos com recursos limitados e polivalentes e, depois de se destacarem os principais objetivos e contributos desta tese, apresenta-se uma revisão da literatura da especialidade.

O problema particular alvo de estudo nesta tese é detalhadamente descrito sendo apresentados os aspetos específicos que o caracterizam. Revisita-se uma formulação matemática de tempo contínuo (*continuous-time formulation*), apresenta-se um exemplo do problema e discutem-se alguns aspetos relacionados com a determinação de soluções admissíveis para o mesmo. Este último tópico é de uma grande relevância quando se lida com problemas que combinam a alocação de recursos polivalentes com o sequenciamento de atividades de um projeto.

De forma a avaliar devidamente a qualidade das soluções obtidas pelos métodos propostos nesta tese, tornou-se necessário o desenvolvimento de um gerador de instâncias para construir um conjunto de instâncias de dimensões maiores do que as que existiam na literatura.

Após apresentar formalmente esse gerador de instâncias, descrevem-se em detalhe as características dos dois conjuntos de instâncias considerados nas experiências computacionais a serem realizadas.

Nas secções seguintes do documento apresentam-se e descrevem-se pormenorizadamente as metodologias desenvolvidas no âmbito desta tese.

Várias formulações matemáticas são estudadas, duas das quais são propostas pela primeira vez nesta dissertação. Através da avaliação da capacidade das formulações estudadas em determinar tanto soluções admissíveis e, possivelmente ótimas, como bons limites inferiores (decorrentes das suas relaxações lineares) para o problema, tornar-se-á claro que as formulações matemáticas de tempo discreto (*discrete-time formulations*) são as que permitem obter os limites inferiores de melhor qualidade enquanto que uma formulação matemática de tempo contínuo mostrou ser a mais adequada para resolver instâncias do problema. Esta tendência é observada para ambos os conjuntos de instâncias considerados. Dois métodos construtivos para obtenção de limites inferiores propostos para o problema de sequenciamento de projetos com recursos limitados são generalizados para considerarem a presença de competências tanto nos recursos como nos requisitos das atividades. Os resultados computacionais mostram que estes métodos melhoram os limites inferiores obtidos pelas formulações matemáticas estudadas em algumas instâncias.

Os problemas reais de sequenciamento de projetos envolvem geralmente um grande número de atividades e de competências. Por isso, a utilização de métodos exatos, como as técnicas comuns para lidar com os modelos matemáticos referidos anteriormente, é frequentemente inviável. Quando confrontado com este tipo de situações, um gestor de projetos pode considerar preferível a obtenção de uma boa solução admissível, não necessariamente uma solução ótima, num tempo aceitável, mediante a utilização de um método aproximado.

Uma reflexão profunda sobre o problema analisado nesta tese revela que este possui algumas características que não estão presentes em alguns casos particulares do mesmo muito estudados, nomeadamente a noção de competência—recursos polivalentes e requisitos das atividades expressos sob a forma de número de recursos necessário para processar cada competência requerida à sua execução.

Assim, com o objetivo de desenvolver métodos aproximados para calcular soluções admissíveis para o problema que explorem melhor as suas características específicas, introduzem-se dois novos conceitos: agrupamento de atividades e atribuição de pesos aos recursos. Os amplamente conhecidos esquemas de geração de sequenciamentos em paralelo (*parallel scheduling scheme*) e em série (*serial scheduling scheme*), propostos inicialmente para o problema de sequenciamento de projetos com recursos limitados, são generalizados para o problema em análise, e os dois conceitos referidos anteriormente são incorporados nestes dois novos procedimentos. Estes algoritmos utilizam as conhecidas regras de prioridade para definir a ordem pela qual as atividades são escolhidas para serem sequenciadas bem como regras de atribuição de pesos para determinar um conjunto de recursos que satisfaça os requisitos de todas as atividades a serem sequenciadas num dado momento com o custo (peso) total mínimo. Seguidamente, duas heurísticas que utilizam os referidos esquemas de geração de sequenciamentos são propostas: uma heurística *multi-pass* que utiliza o referido esquema de geração de sequenciamentos em paralelo e um algoritmo genético de chaves aleatórias viciadas. O cálculo de uma solução admissível utilizando o conceito de sequenciar o projeto do fim para o começo (*backward planning*) também é considerado em ambos os métodos.

A heurística *multi-pass* devolve a solução associada à menor duração do projeto depois de realizar um número específico de execuções (*passes*), cada uma delas associada a uma combinação única de regras de atribuição de prioridade para as atividades, de regras de atribuição de pesos para os recursos, e de redes de precedência (sequenciamento das atividades do começo para o fim do projeto e do fim para o começo).

O algoritmo genético de chaves aleatórias viciadas é uma metaheurística cuja estrutura cromossómica desenvolvida codifica informação relativa a: (i) valores de prioridade das atividades; (ii) os pesos dos recursos; (iii) a forma como um cromossoma é descodificado, nomeadamente o algoritmo de geração de um sequenciamento para o projeto e a rede de precedência a serem utilizados para calcular o tempo associado de conclusão do projeto. Ao incorporar toda esta informação nos cromossomas, é possível tirar partido da estrutura evolutiva do algoritmo genético de chaves aleatórias viciadas que permite a evolução desses da-

dos (alteração nos seus valores) no decorrer da sua execução, em direção a soluções associadas a menores durações do projeto. Consideram-se três variantes do algoritmo genético de chaves aleatórias viciadas de acordo com o tipo de geração de sequenciamento para o projeto a ser utilizado para descodificar os seus cromossomas: (i) todos os cromossomas são descodificados utilizando o *parallel scheduling scheme*; (ii) todos os cromossomas são descodificados utilizando o *serial scheduling scheme*; (iii) o esquema de geração de um sequenciamento para o projeto depende do valor do parâmetro associado.

A heurística *multi-pass* desenvolvida demonstrou ser um método eficiente para a obtenção de soluções admissíveis de qualidade aceitável dentro de um tempo computacional reduzido enquanto que o algoritmo genético de chaves aleatórias viciadas demonstrou ser um algoritmo robusto e uma abordagem aproximada mais competitiva no que diz respeito à determinação de soluções admissíveis de melhor qualidade, em particular para as instâncias mais difíceis como aquelas de média e grande dimensão.

Esta tese termina com um resumo do trabalho desenvolvido e com algumas direções onde pode ser prosseguida a investigação nesta área tanto ao nível de desenvolvimento de metodologias como de algumas generalizações potencialmente interessantes do problema estudado.

Palavras-chave: Sequenciamento de projetos; recursos polivalentes; modelos de otimização; heurísticas; limites inferiores.

Acknowledgments

This dissertation results from a major endeavor undertaken throughout my Ph.D. journey which would had never been possible without the support and encouragement that I received from numerous people to whom I would like to express my sincere acknowledgments.

Foremost, I would like to express my deepest gratitude to my Ph.D. supervisors Professor Francisco Saldanha da Gama of the Faculty of Sciences at the University of Lisbon and Professor Isabel Correia of Faculty of Sciences and Technology at NOVA University whose expertise and immense knowledge along with the continuous feedback provided kept me motivated and steered in the right direction towards the achievement of the outlined objectives.

I would like to express my sincere appreciation for the endless support that my family has provided me throughout my entire life and in particular during the course of my Ph.D. studies.

I am deeply grateful to my grandparents. Thank you for always being a source of inspiration to overcome all the challenges I have faced throughout my life and to contribute to my personal growth by sharing your passions, experiences and discoveries.

I would like to express my heartfelt indebtedness to my parents and my girlfriend for their wise and invaluable advice and for being always extraordinarily supportive and understanding. Without his encouragement and unconditional support, this dissertation would never have seen the light of day.

Finally, I would like to thank all of my friends for the vigorous support provided throughout this Ph.D. journey.

"*You can't be perfect, but if you don't try, you won't be good enough*"

— PAUL HALMOS

Contents

List of Figures

List of Tables

List of Algorithms

Terminology

A	set of pairs of activities having no precedence relations.
AOA	activity-on-arc network.
AON	activity-on-node network.
BRKGA	biased random-key genetic algorithm.
DDT	disaggregated discrete-time precedence constraints, originally proposed by Christofides et al. (1987) for the RCPSP.
E	set of pairs of activities having an immediate precedence relation.
EF_j	earliest finish time of activity j.
ES_j	earliest start time of activity j.
$G_{W_t} = (V_{W_t}, E_{W_t})$	network for determining whether a set of activities can be processed simultaneously by the resources available at time t.
$\widetilde{G}_{W_t} = (V_{W_t}, E_{W_t})$	network obtained from $G_{W_t} = (V_{W_t}, E_{W_t})$ by replacing the weight of each arc (v_0, k), $k \in \mathcal{Z}_{W_t}$ by the corresponding resource weight, w_k.
GA	genetic algorithm.
GRPW	greatest rank positional weight priority rule: activities are sorted in a non-increasing order of the sum of their processing times with the processing times of their immediate successors.
*GRPW**	greatest rank positional weight* priority rule: activities are sorted in a non-increasing order of the sum of their processing times with the processing times of all their successors.
K	total number of resources.
L	total number of skills.
$\mathcal{L} = \{1, \ldots, l, \ldots, L\}$	set of skills.
$\mathcal{L}_j \subseteq \mathcal{L}$	set of skills required by activity $j \in V \setminus \{0, n+1\}$.
$\mathcal{L}^k \subseteq \mathcal{L}$	set of skills mastered by resource $k \in \mathcal{R}$.
$\mathcal{L}_{W_t} = \bigcup_{j \in W_t} \mathcal{L}_j$	set of skills required to process all the activities in W_t.
LF_j	latest finish time of activity j.
LFT	latest finish time priority rule: activities are sorted in a non-decreasing order of their latest finish times.
LP	linear programming.
LPT	longest processing time priority rule: activities are sorted in a non-increasing order of their processing times.

LS_j	latest start time of activity j.
LST	latest start time priority rule: activities are sorted in a non-decreasing order of their latest start times.
makespan	completion time of the project.
$MCNFP(\widetilde{G}_{W_t})$	min-cost flow problem on $\widetilde{G}_{W_t}$.
MILP	mixed-integer linear program.
MIS	most immediate successors priority rule: activities are sorted in a non-increasing order of their number of immediate successors.
MRS	modified resource strength.
MSRCPSP	multi-skill resource-constrained project scheduling problem.
MTS	most total successors priority rule: activities are sorted in a non-increasing order of their total number of successors (immediate and transitive).
n	total number of non-dummy activities to be scheduled.
NC	network complexity.
p_j	processing time of activity $j \in V$.
P_{CT}	continuous-time formulation for the MSRCPSP proposed by Correia et al. (2012).
P_{DT}	discrete-time formulation for the MSRCPSP proposed in this thesis.
P_{DDT}	formulation obtained from P_{DT} by replacing the precedence constraints with the *DDT* ones.
P_M	discrete-time formulation proposed by Montoya et al. (2014).
P_{MDDT}	formulation obtained from P_M by replacing the precedence constraints with the *DDT* ones.
P_N	discrete-time formulation for the MSRCPSP proposed in this thesis.
P_{NDDT}	formulation obtained from P_N by replacing the precedence constraints with the *DDT* ones.
$Pred(j)$	set of immediate predecessors of activity j, $j \in V \setminus \{0\}$.
$\overline{Pred}(j)$	set of all predecessors of activity j (including those obtained by transitivity), $j \in V \setminus \{0\}$.
PSS	parallel scheduling scheme.
pv_j	priority value of activity $j \in V \setminus \{0, n+1\}$.
$\mathcal{R} = \{1, \ldots, k, \ldots, K\}$	set of resources.
$\mathcal{R}_j = \{k \in \mathcal{R} : \mathcal{L}_j \cap \mathcal{L}^k \neq \emptyset\}$	set of resources mastering at least one skill required to process activity $j \in V$.
$\mathcal{R}^l = \{k \in \mathcal{R} : l \in \mathcal{L}^k\}$	set of resources mastering skill $l \in \mathcal{L}$.
r_{jl}	number of resources mastering skill $l \in \mathcal{L}_j$ required to process activity $j \in V$.
$r_{W_t l} = \sum_{j \in W_t} r_{jl}$	number of resources required to fulfill the requirements of each skill $l \in \mathcal{L}_{W_t}$.
RCPSP	resource-constrained project scheduling problem.
RKGA	random-key genetic algorithm.
SbCB	skill-based capacity bound.
SbCSLB	skill-based critical sequence lower bound.
SF	skill factor.

SPT	shortest processing time priority rule: activities are sorted in a non-decreasing order of their processing times.
SSS	serial scheduling scheme.
$Succ(j)$	set of immediate successors of activity j, $j \in V \setminus \{n+1\}$.
$\overline{Succ}(j)$	set of all successors of activity j (including those obtained by transitivity), $j \in V \setminus \{n+1\}$.
UB	upper bound on the makespan of the project.
UV	set of unscheduled activities at some time instant during the execution of an algorithm.
$V = \{0, \ldots, i, \ldots, j, \ldots, n+1\}$	set of activities to be executed. Activities 0 and $n+1$ are dummy; they only represent the beginning and the end of the whole project, respectively.
$V^l = \{j \in V : l \in \mathcal{L}_j\}$	set of activities requiring skill $l \in \mathcal{L}$.
$V_k = \{j \in V : \mathcal{L}_j \cap \mathcal{L}^k \neq \emptyset\}$	set of activities requiring skills mastered by resource $k \in \mathcal{R}$.
w_k	weight of resource $k \in \mathcal{R}$.
W_t	subset of *UV* consisting of activities whose predecessors have all been finished at time t. This set also contains the activities that have no predecessors.
$\mathcal{X}_{W_t l} \subseteq \mathcal{Z}_{W_t}$	set of resources in the optimal solution of $MCNFP(\widetilde{G}_{W_t})$ that meet the skill requirements $l \in \mathcal{L}_{W_t}$.
$\mathcal{Z}_{W_t}$	set of resources that are available at time t and that master at least one skill required by at least one activity in W_t.
$\lceil x \rceil$	the smallest integer greater than or equal to x.
$\lfloor x \rfloor$	the largest integer less than or equal to x.
$\lvert X \rvert$	the cardinality of set X.

CHAPTER 1

Introduction

Project scheduling is one of the most important tasks in project management. It can be defined as the process of sequencing the activities of a project without violating any of the imposed constraints (e.g., precedence relations, resource capacity, skill requirements, budget) such that a predefined performance measure is optimized. The makespan (i.e., the completion time of the project) and total cost are among the most studied performance measures.

The first project scheduling problem studied considered a project comprising a set of activities linked by precedence relations. Each activity, alternatively referred to as task with exactly the same meaning, is associated with a specific processing time. The precedence relations between each pair of activities can be depicted in an activity-on-arc (AOA) network. The goal is to find an optimal schedule by sequencing the activities while satisfying the precedence constraints such that the makespan of the project is minimized. The optimal solution to this problem consists in finding the longest path between the initial and the terminal nodes in the above-mentioned precedence network, which is something that can be achieved in polynomial time. This problem was tackled by Kelley Jr. and Walker (1959) and Malcolm et al. (1959), who developed the well-known Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT), respectively. The main difference between such approaches relies on how the processing times of the activities are determined. More specifically, the CPM assumes that the activities have a deterministic processing time while the PERT uses a probability distribution for computing the processing times of the activities. By assuming resource availability to be unconstrained, a huge drawback associated with the difficulty in properly handling and allocating resources to concurrent activities was reported by Kelley Jr. and Walker (1959) in a practical application of the procedure developed therein.

When resources and their scarceness are taken into account, we dive into a major field of research which has been receiving an increasing attention from both researchers and practitioners that is the class of resource-constrained project scheduling problems (RCPSP). In these problems, resources are limited. Each resource has a specific capacity and is associated with meeting a specific need (e.g., bricks, people, money). Each activity requires specific units of each resource type for being processed. The goal is to find a schedule for the project such that all resource and precedence constraints are satisfied and a predefined performance measure is optimized. Besides the makespan minimization, other objectives have been considered, the more common being related to the presence of resources and to the cost that they represent to the project. This combinatorial optimization problem was proved to be $\mathcal{NP}$-hard in the

strong sense by Błażewicz et al. (1983). The problem has been widely studied in the literature along with many variants of it due to their potential for application in many real-world settings.

Scheduling problems in general have always been tied to the development of new and more powerful computers which has made possible to take the research in this area one step further. The advances in both Computer Science and technology provided the opportunity for developing and studying numerous extensions of the RCPSP, with a particular emphasis to the multi-mode generalizations.

The multi-mode resource-constrained project scheduling problems assume that each activity may be processed in one of multiple ways, the so-called "modes". Each mode of executing a given activity is defined by specific resource requirements and possibly by a specific processing time. The objective is to assign a mode and a start time to each activity, ensuring that the resource requirements and the precedence constraints are satisfied and such that a predefined performance measure is optimized. The most frequently considered performance measures are the same ones described above for the RCPSP.

The class of RCPSP has attracted the attention of numerous researchers who have presented new variants of the basic problem and developed exact and heuristic procedures to address them. For overviews on resource-constrained project scheduling problems, the reader should refer to Herroelen et al. (1998), Brucker et al. (1999), Hartmann and Briskorn (2010), and Węglarz et al. (2011) as well as to the references therein.

More recently, and with the objective of better capturing the characteristics of realistic scheduling problems, a new class of problems emerged—the multi-skill resource-constrained project scheduling problems (MSRCPSP). These problems extend the well-known RCPSP and hence fall into the class of $\mathcal{NP}$-hard combinatorial optimization problems. In a MSRCPSP, the resources besides being available in limited amounts are also associated with the ability of performing a range of functions, each of which associated with mastering a specific skill. These multi-competent resources are referred to in the literature as multi-skill or flexible resources. In this context, the activities may need several skills for their execution, and for each one of those skills, several resource units may be required. These problems are drawing an increasing attention of researchers and practitioners due to their numerous applications in the real-world (e.g., software developments, civil engineering projects) particularly when human resources are involved. In this situation, when the capacity of the resources is unitary, the resulting problem can be formulated as a unit-capacity multi-skill resource-constrained project scheduling problem.

This is the type of problems studied in this thesis. This Ph.D. dissertation falls within the scope of Operations Research and emerges from the need of developing effective solution methodologies for tackling a particular unit-capacity multi-skill resource-constrained project scheduling problem, namely instances of the problem of large dimensions. More particularly, the contributions of this work are as follows (listed in order of appearance):

- a literature review. Due to the importance of discussing the particular problems investigated so far as well as of highlighting the limitations of the solution methodologies adopted, the literature considered to be the most relevant is reviewed;
- an instance generator. The development of an instance generator is of major relevance particularly in situations where no benchmark instances are available. A set of instances of large dimensions was built using such generator. By combining these new instances with a set of smaller instances from the literature it was possible to assess more properly the performance of the methodologies proposed in this thesis, than if only the smaller instances were considered;
- two optimization models. It is of particular interest to develop alternative mathematical formulations for the problem being studied even though it belongs to the class of $\mathcal{NP}$-hard combinatorial optimization problems. In fact, mathematical formulations can still be used both to solve small instances of the problem and to yield lower bounds stemming from their respective linear programming relaxations. Lower bounds are especially useful to evaluate the performance of approximate

methods;

- two methods for computing lower bounds. These methods, which require a negligible computational effort, are specifically useful when, for example, the linear programming relaxations of some mathematical models cannot be solved to optimality within an admissible computational time;
- two schedule generation schemes. These schemes extend well-known methods proposed for the RCPSP to our problem setting. They can be considered the backbone for developing approximate solution methodologies to the problem and hence their development is of major relevance;
- a multi-pass heuristic. When a schedule generation scheme is available, it is possible to perform several executions (multiple passes) of it. Each execution (pass) is associated with both a distinct set of rules (e.g., activity priority rules) among those considered and the schedule generation scheme to be used. By performing multiple passes, instead of a single one, several and possibly distinct makespan values are achieved. The multi-pass heuristic retrieves the best solution which corresponds to the smallest of those makespan values;
- a metaheuristic, namely a biased random-key genetic algorithm. Schedule generation schemes may also be embedded into more sophisticated frameworks. The main reasons for developing this particular genetic algorithm instead of any other metaheuristic are mainly associated with both the high-quality results it provided for resource-constrained project scheduling problems and the fact that the chromosomes resulting from crossover operations always encode a feasible solution to the problem, which is associated with the use of random-keys. This second aspect has a positive impact in the running time of the algorithm since it does not require any additional computational time to recover feasibility. Besides the aforementioned specific features, this metaheuristic also benefits from the general characteristics of genetic algorithms which contribute to an evolution towards better solutions, such as: (i) the ability to escape from local optima; (ii) the ability to handle multiple solutions (under the form of chromosomes) simultaneously; (iii) the ability to adapt problem-related values over time rather than using predefined rules for computing them.

The remainder of this thesis is organized as follows. Chapter 2 presents a literature review of the field of research of multi-skill resource-constrained project scheduling problems. In Chapter 3, the specific problem being studied is thoroughly described and the developed instance generator is presented in detail. Chapter 4 is devoted to the development of mathematical formulations and lower bounds. In Chapter 5, the proposed approximate methods are presented and comprehensively discussed. The computational results of the proposed methodologies are also presented in this chapter. Finally, Section 6 presents some concluding remarks and suggests some directions for further research.

CHAPTER 2

Literature Review

In this chapter, we present a literature review on the relevant research done in the realm of project scheduling problems with multi-skill resources. While reviewing such work, it became clear that the references could be partitioned into two categories according to the efficiency levels at which the skills can be performed by the resources. Such classification allowed the definition of two types of resources: homogeneous and non-homogeneous. In Section 2.1, we present the studies that consider that each skill is performed with the same efficiency by all the resources that master it—homogeneous resources. In Section 2.2, we review the references that assume that the efficiency level at which each skill is performed may not be the same for all resources that master it—non-homogeneous resources. Furthermore, these efficiency levels may be constant or vary over time. We conclude this chapter by presenting an overview of the analyzed research.

2.1 Homogeneous Resources

In this section, we present the studies that consider that all resources mastering a given skill perform it at the same efficiency level. The problems investigated in those studies may be partitioned into two classes: (i) the problems that consider that each activity requires only one resource to meet the demand of each skill needed for its execution—unitary skill requirements; (ii) the problems that do not impose such limit on the skill requirements of the activities—general skill requirements.

We begin by introducing the reviewed research that consider that the skill requirements of the activities are unitary. Then, we present the studies whose analyzed problems do not impose such restriction.

Unitary skill requirements

Li and Womer (2009) investigate a problem in which the activities are linked by generalized precedence relations including minimal and maximal time lags and due dates. The minimal (maximal) time lag is defined for a pair of activities as the least (most) amount of time that has to elapse between their start times. Time lags are used to depict situations where the start time of one activity is constrained to a specific state of completion of some other activity, hence generalizing the classical precedence relations (e.g., start-to-start, finish-to-start).

In this problem, the activities require only one resource unit to fulfill the demand of each one of the skills needed for their execution. The resources have a predefined maximum workload capacity.

The objective is to minimize the total cost associated with the resources without violating a predefined deadline for the project. The authors develop a hybrid Benders decomposition approach and a cut-generating scheme, which links the master problem associated with the resource assignment, i.e., the problem of meeting the skill requirements of the activities, and the subproblem, which deals with the activity scheduling, i.e., the problem of finding whether a specific resource assignment corresponds to a feasible schedule that satisfies the stipulated precedence constraints. The resource assignment problem is formulated as a mixed-integer linear programming formulation (MILP) and solved by branch-and-bound and branch-and-cut whereas constraint programming is applied to check the feasibility of the scheduling problem. When performing the computational experiments, the authors assume that every resource receives the same salary. Hence, the objective of the problem reduces to minimizing the number of distinct resources selected to perform the project. The computational results reveal that the proposed approach appears to be competitive to tackle the specific problem studied. Nonetheless, the characteristics of the instances considered are not fully described.

Dhib et al. (2015) study a project scheduling problem with multi-skill resources where some activities may be preempted, i.e., the execution of some activities may be interrupted. Resources are unavailable within some time-windows and the activities require only one resource for executing each skill needed for their execution. The processing time of each activity is not directly stipulated. Instead, there is a fixed duration associated with the execution of each one of the skills needed to process such activity. Moreover, despite the start time of an activity being the same for all its required skills, in an event of preemption, these skills may be restarted at different time instants. The resources assigned to each pair (activity, skill) cannot be changed even in the event of a preemption of the skill they were assigned to perform. As usual, it is only after all skills have been processed for a given activity, that its immediate successors become eligible to be scheduled. In this problem, each pair (activity, skill) can be seen as a separate task, if we assume that, for each activity, all its associated tasks have the same start time. The authors formulate the problem as a discrete-time MILP optimization model and suggest a parallel scheduling scheme for computing feasible solutions. A parallel scheduling scheme is a time-incrementing schedule generation scheme. A general parallel scheduling scheme can be briefly summarized as follows. Initially, the activities are ranked according to some criteria and a time counter is set to zero. Then, at each time instant that corresponds to the conclusion of some activity(ies), a set of activities whose predecessors have already finished at that time is built—decision set. The highest ranked activity (according to the defined criteria) from that set is iteratively scheduled and the necessary resources are assigned to it. When all activities in that set have been scheduled or no more activities belonging to that set can be scheduled due to the lack of resources available at that time, the time counter is incremented to the minimum finish time of all activities already scheduled. At that time, a new decision set is then built and a new iteration starts. This process is repeated until all the activities in the project have been scheduled. The authors noted that the MILP model is only suitable for dealing with small instances of the problem. Regarding the proposed approximate method, it is important to notice that, by scheduling and fulfilling the requirements of the activities one at a time, a less efficient resource allocation may be obtained. Despite referring to solving a min-cost flow for resource allocation, the authors present neither a cost function nor rules for cost attribution. The computational experience performed is rather superficial.

General skill requirements

Néron (2002) studies a makespan minimization project scheduling problem with multi-skill resources. In this problem, the activities may require more than one resource to meet the demand of each skill needed for their execution. In that work, two methods for deriving lower bounds are proposed: one is based on linear programming and the other one on energetic reasoning. The former is an adaptation of a linear programming approach presented by Carlier and Néron (2003) (whose work was submitted for publication in 2001) for the resource-constrained project scheduling problem and the latter has been

previously applied by Baptiste et al. (1999) for the cumulative version of such problem. The cumulative resource-constrained project scheduling problem assumes that: (i) the capacity of each resource can be greater than one; (ii) each resource may process more than one activity at a time as long as its per period capacity is not surpassed. The energetic reasoning aims at finding, for each time interval, whether the available resources are enough to meet the requirements of all the activities being processed in such interval or not. Both methods for finding lower bounds are classified as destructive since the value of the lower bound supplied as input to the referred methods is incremented until no infeasibility on the constructed schedule is detected. No computational experience on the proposed lower bounds is presented.

Bellenguez-Morineau and Néron (2007) tackle a project scheduling problem with multi-skill resources. The resources may be unavailable within certain time periods, which are known beforehand. The objective is to minimize the makespan of the project. For solving this problem, the authors propose a branch-and-bound method with a depth first branching strategy and three different branching schemes: (i) activity with maximum slack; (ii) adaptation of the graph of compatibility lower bound proposed by Bellenguez-Morineau and Néron (2005) to the particular case where each is skill is performed with the same efficiency by all resources that master it; (iii) energetic reasoning lower bound proposed in Néron (2002). At each node of the branch-and-bound, an activity is selected and its slack is updated. Each node has two child nodes, one of which corresponds to a decrease of the latest finish time of the associated activity (in a number of time units that corresponds to the smallest integer greater than or equal to half of its slack) and the other to an increase of the earliest start time of that activity (in a number of time units that corresponds to the largest integer less than or equal to half of its slack). A leaf node is reached when all the activities have a zero slack. To assure the feasibility of that node, a fixed-job problem has to be solved. This auxiliary problem helps checking whether the skill requirements of the activities can be met at the designated start times. The authors present a superficial computational experience and conclude that the proposed branching strategies have roughly the same performance. The characteristics of the instances used are not extensively detailed. The authors do not present any indicators to measure the degree of difficulty of the referred instances or any evidence that ensures that they are feasible instances of the problem. In this situation, it becomes impossible to categorize the instances according to some sort of parameter(s) and consequently to properly assess the quality of the proposed solution methodology. The only information provided is the total number of instances considered and the ranges of their number of activities, resources and skills.

Correia et al. (2012) propose a MILP mathematical formulation for solving a project scheduling problem with multi-skill resources. The authors derive several valid inequalities and reduction tests with the objective of enhancing the model by fixing some variables at their optimal values beforehand, hence reducing the search space. Since an upper bound on the completion time of the project was required for some of the valid inequalities above, the authors propose a simple heuristic inspired on the well-known parallel scheduling scheme. The approximate method proposed by Correia et al. (2012) is a two-phase heuristic. The first phase comprehends the construction of a feasible solution and the second phase consists in applying a local search procedure to that solution, in an attempt to improve its quality. The constructive phase is slightly different from the generic parallel scheduling scheme described previously. The proposed constructive procedure schedules blocks of at most three activities at a time. All activities in a block start at the same time and the completion time of a block is equal to the maximum finish time across all activities in that block. A block is built by iteratively selecting and scheduling the unscheduled activity with the most total successors and which has all its predecessors already scheduled and finished. The resources are then assigned to that activity by finding a feasible flow in a specific network, without accounting for the skill requirements of other activities that may still be scheduled at that same time. When either the current block consists of three activities or it is not possible to schedule more activities at that time, the time is incremented to the completion time of that block. After building a feasible

solution to the problem, the local search phase of the heuristic starts. The neighborhood of the proposed local search procedure consists of all solutions that differ from the initial one by swapping exactly two activities belonging to adjacent blocks. From those solutions, the algorithm selects the one associated with the least makespan. Due to the absence of judiciously generated instances in the literature, the authors develop a methodology to build instances for the MSRCPSP, which was inspired in the instance generation for the RCPSP. The authors built instances with different characteristics, that caused different degrees of difficulty to the solver. The computational results show that the proposed methodology is competitive for solving instances of small and medium dimensions.

Montoya et al. (2014) develop a branch-and-price algorithm for tackling a multi-skill resource-constrained project scheduling problem. The objective is to minimize the makespan of the project. A discrete-time formulation is presented along with a column generation procedure, and combined into a branch-and-price. The discrete-time formulation and the branch-and-price algorithm are solved using an off-the-shelf solver. The computational results suggest that the branch-and-price procedure may be more competitive than the discrete-time formulation regarding the number of optimal solutions achieved. Nonetheless, it is important to note that, due to the lack of information regarding the instances considered, similarly to Bellenguez-Morineau and Néron (2007), we cannot conclude about the suitability of the proposed branch-and-price method to tackle some properly generated and well-described instances such as the ones built by Correia et al. (2012).

Correia and Saldanha-da-Gama (2015b) noted several inconsistencies in the formulation of Montoya et al. (2014) and present a corrected version of such discrete-time mathematical model.

Correia and Saldanha-da-Gama (2014) study the project scheduling problem with multi-skill resources investigated by Correia et al. (2012) but consider a different objective function. The objective is to minimize the fixed and variables costs associated with the resources while assuring that the project finishes before a stipulated deadline. More precisely, the fixed costs are associated with the selection of a resource to participate in the project (perform a skill for at least one activity) and the variable costs depend on the makespan value of the derived schedule. Both costs are dependent on the number of skills mastered by each resource (i.e., resources mastering a wider range of skills are costlier). The authors develop a mixed-integer optimization model with a non-linear objective function, which was later linearized through the introduction of a set of continuous variables and additional constraints. With the objective of improving the mixed-integer linear programming formulation, several sets of valid inequalities already considered by Correia et al. (2012) were included in the developed mathematical model. In the computational experience, the authors assess the impact of fixed and variable costs separately and together on a comprehensive set of instances built upon those generated by Correia et al. (2012). The numerical results reveal that the proposed approach is efficient for tackling the referred instances.

Correia and Saldanha-da-Gama (2015a) develop a general modeling framework for a project scheduling problem with multi-skill resources. More precisely, the authors address modeling issues related to the development of a MILP modeling framework that depicts the characteristics of the problems studied by Li and Womer (2009) and by Correia et al. (2012), simultaneously. It is shown that several variables can be fixed in a preprocessing phase and that the valid inequalities proposed by Correia et al. (2012) are suitable to be included in the developed framework.

Alba and Chicano (2007) and Drezet and Billaut (2008) also consider homogeneous resources but with some additional assumptions that allow to distinguish the problems studied by these authors from the references presented previously. In the first work, a project scheduling problem faced by software development companies is studied. The objective is to minimize a weighted average of cost and makespan of the project. Two weights are considered (one for the makespan and other for the cost) for modeling the relative importance of the two objectives. The authors provide a general verbal description of the characteristics of the problem. Each multi-skill resource has a monthly salary and a maximum degree of dedication to the project, i.e., the fraction of its working day where it may contribute to the project. Each

activity needs a set of skills and its requirements are expressed under the form of man-hours per month. Unlike the majority of the research on project scheduling problems with multi-skill resources, Alba and Chicano (2007) consider that the processing times of activities are not known beforehand and also that their requirements for being processed are not skill-specific. In fact, the processing times of the activities depend on the number and dedication level of the assigned resources. It is understood that an activity may be processed by only one resource. Moreover, there does not appear to be an upper limit on the number of resources that can be assigned to each activity. The only constraint is that each skill required by an activity must be mastered by at least one of the resources assigned to it. Hence, as long as there are resources that master each one of the required skills, it seems possible to additionally assign resources whose mastered skills are not required by such activity. These resources contribute to decreasing the processing time of that activity. Since no skill-specific requirements are defined, we may conclude that the authors assume that the resources may contribute with all the skills they master simultaneously to the activities they are assigned to perform. From the example given by the authors, it also seems possible to assign the same resource to concurrent activities. No mathematical formulation for the problem is presented. A genetic algorithm is proposed to tackle the described problem. Despite stating that the resources are available to work overtime and that a highly working resource may increase a specific error rate, the authors appear to have neglected these factors when the proposed algorithm was developed. From the presented computational results, it is not possible to conclude that the proposed metaheuristic is an adequate method for tackling this problem. This is due to its inability to provide feasible solutions consistently and to the fact that neither a gap nor other performance measure is presented, which is usually considered mandatory for a proper evaluation of an approximate method. Additionally, the computational time consumed by the proposed solution methodology is not reported. The instances were generated by the authors using an algorithm that is not fully clear and which appears to allow the addition of precedence relations (between pairs of activities) without checking if their inclusion introduces redundancy in the network.

In Drezet and Billaut (2008), a particular project scheduling problem faced by software companies is studied. The requirements of the activities are not expressed as a number of resources per each skill required. Instead, the authors determine, beforehand, the resources that are eligible to perform each activity (possibly accounting for the skills mastered and needed, respectively). Two types of requirements for executing each activity are defined. Both of them are time-dependent, meaning that the requirements of an activity may not be constant throughout its execution. More specifically, the processing time of each activity can be partitioned into unitary time intervals. Each one of these time intervals is associated with a specific demand of resource units, which is bounded by a minimum and maximum value. Additionally, there are also skill requirements in each time interval, but in a different setting. Similarly to Alba and Chicano (2007), the skill requirements of each activity appear not to be skill-specific and hence are similar to the aforementioned demand of resource units. These skill requirements are also bounded by a minimum (maximum) value, which seem to result from aggregating the minimum (maximum) skill requirements across all skills needed to process such activity, in each time interval. The latter means that the pairs (resource, skill) are not determined for each pair (activity, time interval). The number of assigned resources in each time period does not have any impact in the processing time of the activity, which is a fixed and predefined value. The resources that can be assigned to meet each of the described requirements have to be among the ones defined beforehand as capable of performing such activity. A resource can be assigned to perform an activity for its whole processing time or only for a subset of the time periods when it will be in progress. In the latter case, the two following conditions must hold: (i) the activity cannot be preempted, which means that the resource requirements and skill requirements have to remain satisfied (i.e., if withdrawing a resource results in a violation of any of the lower limits for the requirements defined above, another resource will have to replace the removed one); (ii) there is a maximum number of resource assignments/withdrawals during the execution of the activities that

has to be respected. There is a minimal and a maximal length for the working day of each resource as well as a maximum number of activity changes throughout each working day. The objective of the problem is to minimize the maximum lateness. For each activity, its lateness denotes the deviation, in time units, between the due date of such activity and its actual finish time. The authors present a mathematical formulation for the problem and develop a greedy heuristic and a tabu search to compute feasible solutions to it. From the mathematical formulation, we may conclude that the skill requirements of a given activity in each time interval are met considering all the resources busy within such time interval and which were defined as candidate to perform this activity (regardless if they are actually performing this activity or other concurrent activities). The heuristic is based on a serial scheduling scheme and the tabu search considers swap moves between resources and activities and right or left-shifts of one activity by one or several time periods. The serial scheduling scheme is an activity-incrementing schedule generation scheme whose procedure can be briefly summarized as follows. Initially, the activities are ranked according to some criteria. Then, according to the defined criteria, the best precedence feasible activity is scheduled at the earliest (feasible) time possible and the necessary resources are assigned to it. The method repeats until all activities in the project are scheduled. Both approximate methods allow the violation of some constraints namely the ones related to the length of the working day of each resource, its maximum number of assignments/withdrawals and the ones associated with meeting the skill requirements of the activities. The authors claim to have considered two criteria in the objective function. In this objective function, the total number of violated constraints has to be minimized before the maximal lateness. However, it is not clear how such is assured. The computational study is not fully explained and the numerical experiments are very limited. The authors claim to have fixed the values for some parameters of the tabu search according to the results provided by preliminary tests whose details are not presented (e.g., the ranges of values considered for each parameter). The characteristics of the instances considered for the computational experiments performed are not clearly described. The results reveal that the number of violated constraints decreases after employing the tabu search. No gap measure is computed. Furthermore, the derived solutions are associated with the violation of constraints, which implies that they not consist of feasible solutions to the studied problem.

2.2 Non-homogeneous Resources

In this section, we review the references that consider that the efficiency level at which a given skill is performed may vary across the resources mastering that skill. This non-uniform efficiency/expertise/proficiency of performing a given skill is also referred to in the literature as hierarchical levels of skills (see, for instance, Bellenguez-Morineau and Néron, 2005). We designate hereafter this type of resources as non-homogeneous or alternatively, heterogeneous. Moreover, we would like to point out that these efficiency levels may be constant or may vary over time (corresponding to their improvement or deterioration).

We begin by presenting in Section 2.2.1, the research that considers that the efficiency at which a resource performs a given skill remains unchanged throughout the planning horizon—static levels of efficiency. Later, in Section 2.2.2, we review the papers that allow the possibility of these efficiencies varying over time—dynamic levels of efficiency.

2.2.1 Static efficiency levels

Bellenguez-Morineau and Néron (2005) study a project scheduling problem with multi-skill resources where each skill may be performed at different efficiency levels and the resources may be unavailable in predefined time-periods. A resource can perform each one of the skills it masters at every efficiency level equal or smaller than its top efficiency level for that skill. The objective is to minimize the makespan. The

authors develop two lower bounds. One of the proposed lower bounds is based on finding a maximum-weight independent set in a compatibility graph. We note that the maximum-weight independent set problem is $\mathcal{NP}$-hard and hence this approach may be improper to deal with instances of reasonable dimensions. This specific graph is built by linking all pairs of activities that can be in progress simultaneously (i.e., their processing time-windows overlap and there are enough resources to process them at the same time). The authors claim to have tried solving the maximum-weight independent set problem by an approximate method and by a MILP formulation using an off-the-shelf solver. The other lower bound extends the energetic reasoning lower bound proposed by Néron (2002) to account for the multi-levels of efficiency at which a skill can be performed. No computational results are presented.

Yannibelli and Amandi (2011) investigate a multi-skill project scheduling problem where each activity requires skills that may be performed at different efficiency levels. The objective function of the problem regards the assignment of the set of most efficient resources to the activities underlying a software project. No mathematical formulation for the problem is presented. The authors propose a genetic algorithm that coordinates a serial scheduling scheme for tackling this problem. The number of genes in each chromosome is considered to be two times the number of activities in the project. The chromosome structure of the genetic algorithm comprehends information defining a precedence feasible order by which activities are selected to be scheduled and the list of the specific resources that should be assigned to each one of them. Hence, the information specific to each activity spans across two genes. The proposed serial scheduling scheme is not extensively detailed. It is important to note that the absence of a deadline on the completion of the project may allow the building of schedules where activities are processed one after the other, each of which having always the set of most apt resources assign to it. Such solutions appear to be optimal solutions to the problem according to the designated objective. In the computational experiments, it assumed that the resources only master one skill. The characteristics of the instances are not extensively detailed.

Yannibelli and Amandi (2013) extend the previous work by considering a multi-objective problem that seeks to minimize the makespan of the project besides assigning the most effective set of human resources to the activities. The authors propose a metaheuristic consisting of a multi-objective simulated annealing algorithm embedded into a multi-objective evolutionary algorithm. In each iteration of the evolutionary algorithm, the multi-objective simulating annealing is applied to each solution with the objective of preventing the premature convergence of the evolutionary algorithm. The chromosome structure of the evolutionary algorithm is the same one used in the reference presented above (Yannibelli and Amandi, 2011). The authors state that a serial scheduling generation scheme has been employed to decode the chromosomes. No detailed explanation of the developed methodologies, namely the employed serial scheduling scheme is given. The authors assume, similarly to what was considered by Yannibelli and Amandi (2011), that the resources only master one skill in the computational experiments performed. Moreover, the characteristics of the instances are not properly explained. The authors claim to know the optimal solution of the objectives when addressed individually. Nonetheless, no source for those values of reference is provided.

Multi-skill resources have also been considered in multi-project environments (i.e., when several concurrent projects have to be scheduled) but with additional assumptions. We present next, some of the research done in this direction.

Heimerl and Kolisch (2010) study a cost minimization multi-project scheduling problem with multi-skill resources faced by a company. The authors do not consider a project selection phase, which implies that all the available projects have to be scheduled. For each project, the sequence by which the activities are processed and consequently its makespan is known beforehand. Nonetheless, it is necessary to determine both the start time of each project (which have to be within a predefined time-window) and the resources that should be assigned to meet its skill requirements in each time slot where it will be in execution. There are two kinds of resources: internal and external, which have specific costs. The inter-

nal resources have a workload capacity that is partitioned into regular and overtime, the only difference is the cost associated with each one. The authors stipulate a cost of hiring external resources higher than the overtime cost of the internal resources. Additionally, a limit on the maximum proportion of the work done by external resources is imposed. These assumptions seek to avoid outsourcing all the work. The efficiency level at which each skill is performed may vary across all the internal resources that master it. Conversely, each skill is performed at the same efficiency level by all external resources that master it. A resource may be assigned to multiple projects simultaneously, possibly performing different skills for each one of these projects. Such assumption has also been made by other authors (cf. Alba and Chicano, 2007, Gutjahr et al., 2008 and Gutjahr et al., 2010). The time taken to perform a given skill within a specific time-window is smaller for (internal) resources having higher efficiency levels of that skill. Nonetheless, this only impacts the usage of the regular and overtime capacities of the resources, since the processing time of the activities is fixed (the makespan of the project is known beforehand). The problem is formulated as a mixed-integer linear programming model. The authors further evaluate extensions of the problem by considering: a budget for the project; the scarceness of external resources; the hiring of discrete quantities of external resources (multiples of a given number). The model is enhanced with some constraints aimed at improving the lower bound provided by its linear programming relaxation. The computational results validate the decision of including additional constraints in terms of the derived lower bounds. The authors also study the impact of several factors on the objective function of the problem: the size of the time-windows for the start time of each project; the number of projects; the number of skills mastered by the internal resources; the workload capacity. The authors also assess the impact that a centralized (single pool of resources) and a de-centralized planning (resources are spanned across several departments) have in the cost of the project. They conclude that the centralized planning scheme is the most competitive approach since it tends to render easier problems and to allow a greater assignment efficiency. The computational results reveal that the proposed exact approach (MILP with a tight linear programming relaxation) provides smaller total costs than the two simple heuristic methods already being used by the company. The effectiveness of the former is strongly dependent on the size of the time-windows for start of the projects, which the authors state that have to be "modest", which may be understood as rather small, since they have a great influence in the tightness of the linear programming relaxation of the proposed MILP. The referred approximate methods proved to be unable to provide feasible solutions for some instances when a minimum ratio of the work to be done by internal resources is imposed.

Kolisch and Heimerl (2012) extend the previous work by considering a disaggregated version of the problem. In this setting, the start time of the activities involved in each project (i.e., the schedule of each project) has to be determined. The activities have to be processed within predefined time-windows and are linked by precedence relations with minimal and maximal time lags. These concepts were already presented when the problem studied by Li and Womer (2009) was reviewed. Each activity has exactly one predecessor. If some activity is selected to be outsourced then all its successors must also be outsourced. A mixed-integer linear programming model is presented along with additional cuts. The authors partition the problem into two subproblems: (i) the scheduling problem, which aims at determining the start times of the activities; and (ii) the staffing problem, which is associated with meeting the skill requirements of the activities. For the former, the authors use a metaheuristic that combines a genetic algorithm and a tabu search and for the latter, a generalized network simplex (GNS) algorithm is used to solve the associated generalized minimum cost network flow problem. The genetic algorithm penalizes infeasible solutions provided by the GNS in an attempt to prevent their propagation into further generations. A tabu search is then applied to the best solution found in each iteration. A neighborhood consisting of modifying the start time of an activity is proposed (the authors point out the need of verifying and correcting the start time of its successors to assure a schedule that is precedence feasible). The computational results show that the proposed metaheuristic is more suitable for computing feasible solutions to this problem than

an off-the-shelf solver. In fact, the solver failed to solve the strengthen MILP formulation (within the imposed time limit) for reasonably sized instances (i.e., 10 projects, 6 activities per project, 10 resources and time-window size per activity of 3 time units).

2.2.2 Dynamic efficiency levels

In some real-world settings, when human resources are involved, it is reasonable to assume that task repetitiveness, academic education and technical training may contribute positively to the development of competences (i.e., learning), which tend to increase the efficiency at which the associated skills are performed. Conversely, it is also true that if a resource refrains itself from performing specific skills for a period of time, it may become less and less proficient in those skills over time (i.e., forgetting), which causes a decrease in the efficiency at which those skills are performed.

Some research considering those aspects (although simplifying others) include Gutjahr et al. (2008) and Gutjahr et al. (2010).

Gutjahr et al. (2008) tackle a multi-project scheduling problem with an *a priori* project selection phase. The objective is to maximize a weighted average of the economic gains from the projects and the increments in the competences of the resources. Similarly to Heimerl and Kolisch (2010), there are no decisions regarding the start times of the activities. The skill requirements of each project are predetermined within the predefined time-windows for processing the activities. The resources have non-homogeneous efficiencies for each skill mastered, and those efficiencies may be dynamic, in the sense that they may improve or deteriorate—the so-called learning and forgetting effects. Similarly to the aforementioned studies, namely Alba and Chicano (2007), Heimerl and Kolisch (2010) and Kolisch and Heimerl (2012), a resource can contribute with multiple skills to concurrent activities as long as its capacity is not surpassed in each period. The authors develop a non-linear mixed-integer programming formulation for the problem and show that the referred mathematical model can be approximate by a MILP when specific problem settings are considered. However, unlike Heimerl and Kolisch (2010) and Kolisch and Heimerl (2012), it is not possible to hire external resources. A two-staged approximate approach is derived. It consists of a greedy heuristic, which is used for the scheduling and staff assignment problem, and two metaheuristics (ant colonization optimization and genetic algorithm), which are used for project portfolio selection. The decision concerning the selection of the projects that will constitute a certain portfolio is made at the beginning and remains unchanged in the course of the algorithm. The start time of each project is considered to be equal to the earliest start of all its integrating activities. Hence, we may consider that the proposed greedy heuristic does not deal directly with the scheduling of the projects but only with meeting their requirements. Moreover, this procedure seems not to account for the learning and forgetting of the skills mastered by the resources. Additionally, the authors claim that this greedy heuristic may retrieve infeasible solutions and that in such event the metaheuristic selects a different project portfolio. In the computational experience, the authors compare the performance of their approximate method with a simplified version of the proposed MILP formulation, which is solved using an off-the-shelf solver. It is concluded that the genetic algorithm is the most appropriate method for project portfolio selection among the two metaheuristics tested.

Gutjahr et al. (2010) study a bi-objective generalization of the problem reviewed above. Two metaheuristics are proposed for selecting the projects, namely a nondominated sorting genetic algorithm and a pareto ant colony optimization method, which consist of generalizations of the methods applied in Gutjahr et al. (2008). After selecting a project portfolio and calculating the value of the associated objective (economic gains) a multi-objective resource assignment problem has to be solved. The authors simplify such continuous assignment problem by considering a unique objective and formulate it as a single objective linear programming model, which is solved using an off-the-shelf solver. The authors evaluate the performance of the proposed metaheuristics on two different sets of instances, one of which

consisting of real data while the other is associated with artificial created problems. None of the referred sets of instances is described in detail. It is concluded that the pareto ant colony optimization method achieves the best solutions for the former and the nondominated sorting genetic algorithm provides the best results for the latter set of instances.

2.3 Conclusions

In this chapter, we have reviewed the most relevant literature on project scheduling problems with multi-skill resources. The area of project scheduling problems with multi-skill resources is a recent field of research. However, we observe that a lot of effort has already been put into developing methodologies to deal with various types of these problems. This review is of particular interest since, to the best of the author's knowledge, no review paper focused on this specific class of problems has been published so far.

The references were partitioned into two categories according to the type of resources involved: homogeneous—each skill is performed with the same efficiency by all resources that master it; non-homogeneous—the efficiency level at which a skill is performed may vary across the resources that master it.

Let us first notice that the class of problems involving homogeneous resources has been receiving the most attention. Nonetheless, a closer look at the proposed solution methodologies suggests that the development of effective approximate methods has not been extensively studied for these problems. In fact, aside from Dhib et al. (2015), which tackles a preemptive version of the problem and Alba and Chicano (2007) and Drezet and Billaut (2008), which make additional assumptions, only Correia et al. (2012) have proposed a constructive heuristic to the original setting of the problem. These approximate methods are either not clearly described, with the exception of Correia et al. (2012), or seem to disregard some fundamental features of the project scheduling problems with multi-skill resources (e.g., the notion of skill and the possible scarceness of resources mastering certain skills). Such insights reinforce the need to develop effective methods for computing upper bounds for these problems, which is one of the main goals of the present thesis. Furthermore, we observe that all the reviewed references dealing with homogeneous resources, study a problem comprising a single project.

If we look closely at the research considering non-homogeneous resources, we observe that a lot of effort has been put into developing methodologies to tackle a wide range of these problems, with a particular emphasis to those considering multiple projects. In these problems, each skill may be performed and required at different levels of proficiency by the resources and by the activities, respectively. This increased complexity tends to make these problems intractable at their original setting, hence the proposed solution methodologies often end up tackling a simplified version of the initial problem.

The absence of benchmark instances and the general lack of detail and rigor in the description of the instances considered in the computational experiments performed by some authors constitute a major drawback in what regards the proper evaluation and comparison of the proposed methodologies, which are also often not clearly described. Hence, it is of particular interest to develop and formally present an instance generator and to determine a set of parameters that clearly define the characteristics of each (sub)set of instances considered. We observe that only two references have dedicated their work to the development of lower bounds, one of which for the problems having homogeneous resources and the other for the problems considering heterogeneous resources.

We summarize the work developed so far in Tables 2.1 and 2.2. The references are classified according to the characteristics and assumptions of the problems studied therein. In Table 2.1, we present the features of the problems, objectives and methodologies proposed. In Table 2.2, we identify the fundamental aspects that characterize the resources and the activities involved in the investigated problems.

Table 2.1: Literature review: characteristics of the problems and proposed methodologies.

Resources	Skill efficiencies	Reference	Project			Objective			Methodology(ies)		
			Single	Multi	Selection	Single	Multi	Function	Exact	Approximate	LB
Homogeneous	n/a	Néron (2002)	✓			✓		M			✓
		Bellenguez-Morineau and Néron (2007)	✓			✓		M	✓		
		Alba and Chicano (2007)	✓			✓		M + C (w)		✓	
		Drezet and Billaut (2008)	✓			✓		L	✓	✓	
		Li and Womer (2009)	✓			✓		C	✓		
		Correia et al. (2012)	✓			✓		M	✓	✓	
		Montoya et al. (2014)	✓			✓		M	✓		
		Correia and Saldanha-da-Gama (2014)	✓			✓		C	✓		
		Correia and Saldanha-da-Gama (2015a)	✓			✓		M or C	✓		
		Dhib et al. (2015)[1]	✓			✓		M	✓	✓	
Heterogeneous	static	Bellenguez-Morineau and Néron (2005)	✓			✓		M	✓		✓
		Yannibelli and Amandi (2011)	✓			✓		ER		✓	
		Yannibelli and Amandi (2013)	✓				✓	ER + M		✓	
		Heimerl and Kolisch (2010)		✓		✓		C	✓	✓	
		Kolisch and Heimerl (2012)		✓		✓		C	✓	✓	
	dynamic	Gutjahr et al. (2008)[2]		✓	✓	✓		EG + RC (w)	✓	✓	
		Gutjahr et al. (2010)[2]		✓	✓		✓	EG + RC	✓	✓	

C: min. cost; ER: max. efficiency of assigned resources; EG: max. economic gains from projects; L: min. max. lateness; LB: Lower Bound(s); M: min. makespan; RC: max. resources' competences; (w): weighted.

1 Preemption is allowed, moreover each skill may be preempted individually.

2 Projects are not to be explicitly scheduled, since the start time of a project is set equal to earliest start time of all its integrated activities, which is known beforehand.

Table 2.2: Literature review: characteristics of the resources and of the activities.

Resources	Skill efficiencies	Reference	Resources		Activities		
			Unavailability	ACA	NDS	GPC	Outsourcing
Homogeneous	n/a	Néron (2002)					
		Bellenguez-Morineau and Néron (2007)	✓				
		Alba and Chicano (2007)	✓[3]	✓			
		Drezet and Billaut (2008)	✓[3]				
		Li and Womer (2009)	✓[3]			✓	
		Correia et al. (2012)					
		Montoya et al. (2014)					
		Correia and Saldanha-da-Gama (2014)					
		Correia and Saldanha-da-Gama (2015a)	✓[3]			✓	
		Dhib et al. (2015)[1]					
Heterogeneous	static	Bellenguez-Morineau and Néron (2005)	✓				
		Yannibelli and Amandi (2011)					
		Yannibelli and Amandi (2013)					
		Heimerl and Kolisch (2010)		✓	✓		✓
		Kolisch and Heimerl (2012)		✓		✓	✓
	dynamic	Gutjahr et al. (2008)[2]		✓	✓		
		Gutjahr et al. (2010)[2]		✓	✓		

ACA: Assignable to concurrent activities (i.e., in a given time period, a resource may be assigned to perform multiple activities/projects, possibly using different skills); GPC: Generalized precedence constraints; NDS: No decision regarding the start times of the activities.

1 Preemption is allowed, moreover each skill may be preempted individually.
2 Projects are not to be explicitly scheduled, since the start time of a project is set equal to earliest start time of all its integrated activities, which is known beforehand.
3 limited workload capacity.

CHAPTER 3

The studied problem

In this chapter, we present and describe in detail the multi-skill resource-constrained project scheduling problem (MSRCPSP) being investigated in this thesis. This is the problem studied by Correia et al. (2012) whose work constitutes the starting point for this dissertation.

We begin by verbally describing the problem and the specific assumptions that are associated with it in Section 3.1. Then, an example is presented. In Section 3.2, the relevant notation is introduced and a mathematical formulation from the literature is revisited. In Section 3.3, we discuss several problem-specific properties and additional concepts of relevance for the developments to be presented in the following chapters. In Section 3.4, we revisit the instances generated by Correia et al. (2012) and we formally propose a new instance generator, used to build a new set of instances. These two sets of instances will be used for evaluating the performance of the solution techniques proposed in the following chapters. This chapter ends with Section 3.5 with some concluding remarks.

3.1 Problem description

We consider a project comprising a set of activities, a set of resources and a set of skills. The activities are linked by finish-to-start precedence relations, which force an activity not to be started before all its predecessors have finished. Each activity is associated with a processing time and has specific requirements for being executed. The precedence relations between pairs of activities can be depicted in an activity-on-node (AON) network that also contains two dummy activities, i.e., activities with a null processing time and no resource requirements, that represent the start and the conclusion of the project. Apart from the precedence constraints, the activities are also interrelated by resource constraints, which are more complex than the ones considered in the classical RCPSP. Each resource masters one or more skills. The activities require several skills for their execution and, for each one of those skills, one or more resources are needed. A resource can be involved in at most one activity at a time, to which it contributes with only one skill it masters and that is required by such activity. Furthermore, once a resource is assigned with a skill for performing some activity, it remains so for the whole processing time of the activity. The objective of the problem is to minimize the completion time of the project. The decisions to make comprise the start times of the activities and the pairs (resource, skill) that should be assigned to each activity, such that the precedence and resource constraints are satisfied.

This problem belongs to the class of project scheduling problems with multi-skill resources. The

MSRCPSP can be looked at as a specific case of the multi-mode extension of the RCPSP. In fact, if we consider that a mode of executing an activity is defined by a set of pairs (resource, skill) that meet all its skill requirements, the resulting problem is a multi-mode RCPSP with a very high number of modes. Hence, by generalizing the RCPSP, the MSRCPSP being studied is an $\mathcal{NP}$-hard combinatorial optimization problem in the strong sense.

In addition to the description of the problem presented above, we also consider the following assumptions:

- all precedence relations are of type finish-to-start, which means that an activity may only start after all its predecessors have finished.
- No minimal or maximal time lags are considered.
- the processing times of the activities are fixed, positive and integer values (hence independent from the resources assigned to it).
- the setup times are included in the processing times of the activities.
- preemption is not allowed, i.e., once an activity starts being executed, it cannot be interrupted.
- the number of resources needed to process each skill required by each activity is a fixed, positive and integer value.
- the resources are renewable and have a unitary capacity.
- the resources are homogeneous, i.e., all the resources mastering a specific skill, perform it with the same efficiency.
- all the information above is deterministic and known beforehand.

As a result, we can immediately conclude that the makespan of the project is a positive integer less than or equal (when the activities are processed sequentially) to the sum of the processing times of the activities. Typically, it is possible to parallelize the execution of some activities and thus, the minimum makespan is usually lower than that upper limit. We discuss this and other issues later in this document.

Next, we present an example of the problem.

Example 3.1 *Consider a project with 5 activities, 5 resources and 2 skills. The information associated with the activities and the resources is presented in Tables 3.1 and 3.2, respectively. The immediate precedence relations between pairs of activities are depicted in the directed acyclic graph presented in Figure 3.1, which follows an AON representation.*

Table 3.1 consists of 4 columns. Each row corresponds to an activity whose label is depicted in the first column. For each activity, Column 2 indicates its processing time and Columns 3 and 4 refer to its skill requirements in terms of the number of resources required for skill 1 and skill 2, respectively.

Table 3.2 consists of 3 columns. Each row corresponds to a resource whose label is depicted in the first column. Columns 2 and 3 indicate whether a resource masters skill 1 and skill 2, respectively.

From the AON network depicted in Figure 3.1, we observe that activity 3 is the only activity with more than one predecessor and more than one successor. We can already conclude that, according to these precedence relations, activity 3 *cannot be processed in parallel with any other activity. Each arc in this network has a weight equal to the processing time of the activity it departs from. For instance, the weight of the arc (1,3) is equal to 3—the processing time of activity 1. Activities 0 and 6 are dummy activities associated with the beginning and the end of the project, respectively. As referred previously, these activities have a null processing time and no skill requirements.*

Given the data presented above, we conclude that the project lasts at most 9 time units—the sum of the processing times of all the activities, and at least 6 time units, the length of the so-called critical

Table 3.1: Example of the problem: activities.

Activity	Processing time	Skill requirements skill 1	skill 2
1	3	2	3
2	2	0	1
3	1	2	1
4	1	1	2
5	2	1	0

Table 3.2: Example of the problem: resources.

Resource	Skills mastered skill 1	skill 2
1		✓
2	✓	
3	✓	✓
4		✓
5	✓	✓

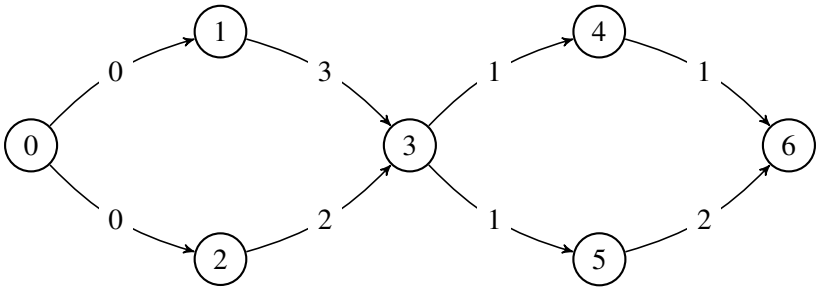

Figure 3.1: Example of the problem: precedence network.

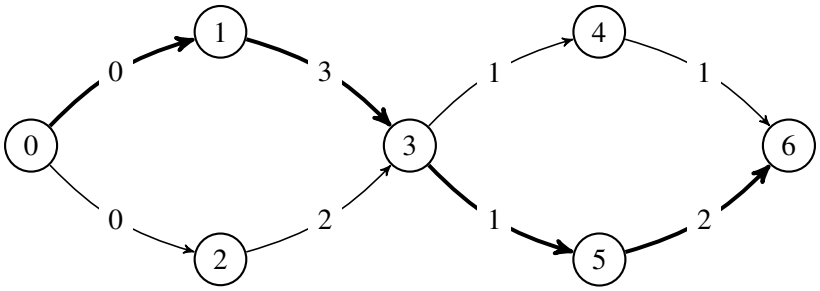

Figure 3.2: Example of the problem: the critical path.

path—the longest path from 0 to 6 in the precedence network, see Figure 3.2 (highlighted in boldface and consisting of activities 1, 3 and 5). In a real-world context, resources are limited, and hence the schedule whose solution corresponds to the length of the critical path may be infeasible due to a violation of at least one resource constraint.

Two or more activities may overlap if none of them is linked to any other of these activities by direct or transitive precedence relations and if the resources available in the time slots where they will be in progress are enough to meet all their skill requirements at the same time. The conditions for a set of activities to overlap are formally presented in Section 3.3.1.

The precedence constraints are easily validated by looking at the network depicted in Figure 3.1 while the resource constraints can be checked by looking at the requirements of the activities (Columns 3 and 4 of Table 3.1) and the skills mastered by the resources (Columns 2 and 3 of Table 3.2).

In this example, we observe that the only activities that are not linked by any precedence relation are the two pairs of activities: 1 and 2, 4 and 5. After isolating those two pairs of activities it becomes only necessary to verify whether the available resources are enough to meet their skill requirements.

Starting with activities 4 and 5, we notice that the skill requirements of activity 4 can be met by assigning resource 2 to perform skill 1, and resources 1 and 3 to perform skill 2. Given the decision associated with the assignment above, we are constrained in terms of selecting resources to process activity 5. Nonetheless, we may allocate resource 5 to perform skill 1 for activity 5, its unique skill demand. A closer look at Tables 3.1 and 3.2 allows to conclude that other resource allocations that also meet the skill requirements of these activities would be possible.

If we apply the same reasoning to activities 1 and 2, we quickly realize that these activities cannot

overlap. In spite of not being linked by precedence relations, there are not enough resources to meet all their skill requirements simultaneously.

We conclude that this solution is associated with a makespan of 8 time units—the sum of the processing times of activities 1, 2 and 3 (processed sequentially) with the maximum of the processing times of activities 4 and 5 (processed in parallel). Such schedule, which consists of an optimal solution to this problem, is presented in Figure 3.3.

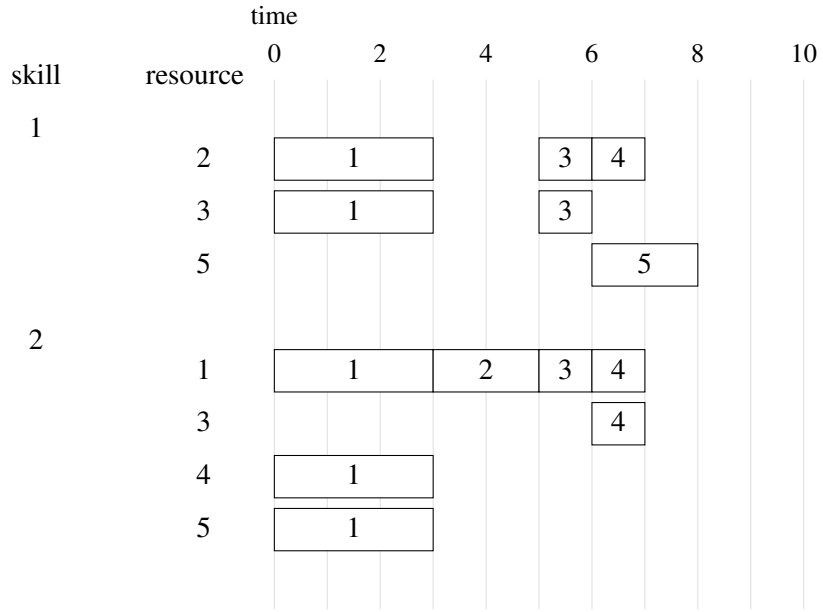

Figure 3.3: Example of the problem: an optimal schedule.

It is possible to derive other optimal solutions from the optimal solution presented by swapping the processing order of some activities, changing their resource assignments or both.

An alternative optimal solution to this problem may be obtained by either swapping the processing order of activities 1 and 2 or meeting the requirements of activity 4 for skill 2 through the assignment of resource 4 instead of resource 3 or performing both changes.

Notice that we carefully selected the resources to meet the skill requirements of activities 4 and 5 simultaneously. In fact, if the requirements of activity 4 would have been met through the assignment of resource 2 to skill 1 and resources 3 and 5 to skill 2, activity 5 could no longer overlap activity 4, since the resources mastering skill 1 (unique skill required by activity 5) were already assigned to activity 4.

The selection and assignment of resources play a major role in what concerns the computation of feasible solutions for this problem, since they have a direct influence in the makespan of the project. Due to the great relevance of this subject, we are debating this and some other issues in the following sections.

3.2 Notation and a mathematical model

In this section, we present some of the relevant notation and terminology that will be considered in the developments presented in the following chapters. We also revisit the MILP model introduced by Correia et al. (2012). As far as the notation is concerned, we introduce the following:

$V = \{0, \ldots, i, \ldots, j, \ldots, n+1\}$	:	set of activities to be executed. Activities 0 and $n+1$ are dummy; they represent the beginning and the end of the whole project, respectively. Their processing time is equal to 0 and they do not have any skill requirements.
$\mathcal{R} = \{1, \ldots, k, \ldots, K\}$	:	set of resources.
$\mathcal{L} = \{1, \ldots, l, \ldots, L\}$	:	set of skills.
$\mathcal{L}_j \subseteq \mathcal{L}$	:	set of skills required by activity $j \in V \setminus \{0, n+1\}$.
$\mathcal{L}^k \subseteq \mathcal{L}$	:	set of skills mastered by resource $k \in \mathcal{R}$.
$Succ(j)$	:	set of immediate successors of activity j, $j \in V \setminus \{n+1\}$.
p_j	:	processing time of activity $j \in V$.
r_{jl}	:	number of resources mastering skill $l \in \mathcal{L}_j$ required to process activity $j \in V$.

We assume that the values of p_j and r_{jl} are positive integers, $j \in V \setminus \{0, n+1\}$, $l \in \mathcal{L}_j$.

Based upon the previous sets and parameters it is also possible to define the following notation:

$V^l = \{j \in V : l \in \mathcal{L}_j\}$	:	set of activities requiring skill $l \in \mathcal{L}$.
$V_k = \{j \in V : \mathcal{L}_j \cap \mathcal{L}^k \neq \emptyset\}$	:	set of activities requiring skills mastered by resource $k \in \mathcal{R}$.
$\mathcal{R}^l = \{k \in \mathcal{R} : l \in \mathcal{L}^k\}$	:	set of resources mastering skill $l \in \mathcal{L}$.
$\mathcal{R}_j = \{k \in \mathcal{R} : \mathcal{L}_j \cap \mathcal{L}^k \neq \emptyset\}$	:	set of resources mastering at least one skill required to process activity $j \in V$.
$Pred(j)$	:	set of immediate predecessors of activity j, $j \in V \setminus \{0\}$.
$\overline{Pred}(j)$	:	set of all predecessors of activity j (e.g., by transitivity), $j \in V \setminus \{0\}$.
$\overline{Succ}(j)$	:	set of all successors of activity j (e.g., by transitivity), $j \in V \setminus \{n+1\}$.
E	:	set of pairs of activities having an immediate precedence relation.
A	:	set of pairs of activities having no precedence relations.
UB	:	upper bound on the makespan of the project.
ES_j	:	earliest start time of activity j, $j \in V$.
EF_j	:	earliest finish time of activity j, $j \in V$.
LS_j	:	latest start time of activity j, $j \in V$.
LF_j	:	latest finish time of activity j, $j \in V$.

The methodological developments to be presented require the definition of the following additional notation:

UV	:	set of unscheduled activities at some time instant during the execution of an algorithm.
W_t	:	subset of UV consisting of activities whose predecessors have all been finished at time t. This set also contains the unscheduled activities that have no predecessors.

$\mathcal{Z}_{W_t}$: set of resources that are available at time t and that master at least one skill required by at least one activity in W_t.

$\mathcal{L}_{W_t} = \bigcup_{j \in W_t} \mathcal{L}_j$: set of skills required to process all the activities in W_t.

$r_{W_t l} = \sum_{j \in W_t} r_{jl}$: number of resources required to fulfill the requirements of each skill $l \in \mathcal{L}_{W_t}$.

Given a list of immediate successors (or a list of immediate predecessors) for each activity, it is possible to represent immediate precedence relations in a direct acyclic AON network $G = (V, E)$ where E is a set of arcs. Each arc in E corresponds to an immediate precedence relation between the two activities it connects. The nodes are numerically labeled in such a way that each activity (i.e., node) is always labeled by a number smaller than the numbers labeling its successors. Two dummy activities are included in the precedence network to mark the beginning and the end of the project. These activities have null processing times and no skill requirements.

For each activity $j \in V$, it is possible to compute its earliest start time, ES_j, and its latest start time, LS_j, in polynomial time by using an algorithm for finding a longest path in the network $G = (V, E)$. For an activity j, the computation of ES_j reduces to determining the earliest precedence-feasible time where activity j can start, that is the length of the longest path connecting the dummy activity 0 with activity j. The earliest finish time EF_j is then directly obtained by adding the value of p_j to the previously computed ES_j. Obviously, we observe that $ES_0 = 0$ and ES_{n+1} is equal to the length of the longest path in G from nodes 0 to $n + 1$ (i.e., critical path).

In order to compute the latest start LS_j and finish times LF_j an upper bound, UB, on the makespan of the project is required. The values of LS_j and LF_j are then obtained by using backward recursion, where LS_j is given by UB minus the length of the longest path that connects activity j with the dummy activity $n + 1$; then LF_j can be set equal to $LS_j + p_j$.

Finally, one additional set is considered, denoted by A, representing all pairs of activities having no direct or transitive precedence relations and that hence can overlap if there are enough resources to meet their skill requirements simultaneously. A direct (transitive) precedence relation is represented by $i \prec j$ ($i \prec\!\!\prec j$) if i is a direct (transitive) predecessor of j.

$$A = \{(i,j) \in V \times V \ : \ i \not\prec j \wedge j \not\prec i \wedge i \not\prec\!\!\prec j \ \wedge \ j \not\prec\!\!\prec i\}$$

Activities are scheduled over time. It is assumed the value 0 for the origin of time. This is a common practice in Project Scheduling since it does not remove generality from the problem. Due to the fact that the processing times of the activities are integer numbers, the start and finish times of the activities are also integer. Hence, hereafter whenever we refer to a "time t", t is an integer.

A MILP model

Correia et al. (2012) proposed a mixed-integer linear programming model for the problem described in Section 3.1. Next, we revisit such model.

Apart from the variables associated with the allocation of the resources, this model makes use of continuous variables representing the start times of the activities as well as binary variables defining the execution sequence for any pair of activities having no precedence relation between them. This type of model is often referred to as a sequence-based or disjunctive model.

The decision variables are the following:

S_j : start time of activity $j \in V$.

$$y_{ij} = \begin{cases} 1, & \text{if activity } i \text{ finishes before activity } j \text{ starts;} \\ 0, & \text{otherwise.} \end{cases} \qquad i,j \in V \wedge (i,j) \in A$$

We note that variables y_{ij} are only defined for pairs $(i,j) \in A$ because for all the other pairs of activities, sequence is automatically defined by the precedence relations.

The following variables handle the multi-skill nature of the resources when they are allocated to the activities:

$$x_{jlk} = \begin{cases} 1, & \text{if resource } k \text{ contributes with skill } l \text{ for activity } j; \\ 0, & \text{otherwise.} \end{cases}$$

$$j \in V, k \in \mathcal{R}_j, l \in \mathcal{L}^k \cap \mathcal{L}_j$$

Considering the decision variables defined above we obtain the following model, that we denote by P_{CT}:

$$\min \quad S_{n+1} \tag{3.1}$$

subject to:

$$S_j \geq S_i + p_i \qquad i,j \in V \wedge (i,j) \in E \tag{3.2}$$

$$S_j \geq S_i + p_i - M\left(1 - y_{ij}\right) \qquad i,j \in V \wedge (i,j) \in A \tag{3.3}$$

$$y_{ij} + y_{ji} \leq 1 \qquad i,j \in V \wedge (i,j) \in A : i < j \tag{3.4}$$

$$\sum_{k \in \mathcal{R}^l} x_{jlk} = r_{jl} \qquad j \in V \setminus \{0, n+1\},\ l \in \mathcal{L}_j \tag{3.5}$$

$$\sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} x_{jlk} \leq 1 \qquad k \in \mathcal{R},\ j \in V_k \tag{3.6}$$

$$\sum_{l \in \mathcal{L}^k \cap \mathcal{L}_i} x_{ilk} + \sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} x_{jlk} \leq y_{ij} + y_{ji} + 1 \qquad k \in \mathcal{R},\ i,j \in V_k \wedge (i,j) \in A \tag{3.7}$$

$$ES_j \leq S_j \leq LS_j \qquad j \in V \tag{3.8}$$

$$x_{jlk} \in \{0,1\} \qquad k \in \mathcal{R},\ j \in V_k,\ l \in \mathcal{L}^k \cap \mathcal{L}_j \tag{3.9}$$

$$y_{ij} \in \{0,1\} \qquad i,j \in V \setminus \{0, n+1\} \wedge (i,j) \in A \tag{3.10}$$

The objective function (3.1) represents the start time of dummy activity $n+1$, which is equal to the makespan of the project (to be minimized). Constraints (3.2) ensure that the precedence relations hold for all pairs of activities $(i,j) \in E$. Constraints (3.3) relate the start times of the activities $(i,j) \in A$ (not having any precedence relation) with the variables y_{ij}. In these constraints M denotes a large value (big-M). These constraints are only relevant to the model when $y_{ij} = 1$ and in that situation it imposes a precedence relation that forces activity j not to be started before activity i is finished. Constraints (3.4) complement constraints (3.3) by stating that for each pair of activities $(i,j) \in A : (i < j)$ either i and j overlap ($y_{ij} = y_{ji} = 0$) or i is finished before j starts ($y_{ij} = 1$ and $y_{ji} = 0$) or j is completed before i starts ($y_{ji} = 1$ and $y_{ij} = 0$). Constraints (3.5) assure that the skill requirements of the activities are fulfilled through the assignment of the necessary resources. Constraints (3.6) state that each resource contributes with at most one skill (that it masters) to an activity (which requires it). Constraints (3.7) limit the assignment of each resource to at most one activity at a time. This restriction implies that these constraints should not be considered for activities having precedence relations $(i,j) \in E$, as these will never overlap in any feasible schedule. If a resource is assigned to both activities i and j then $y_{ij} + y_{ji} \geq 1$, which together with constraints (3.4) assure that $y_{ij} + y_{ji} = 1$ and thus that i and j cannot overlap (either i finishes before j starts or the other way around). Constraints (3.8) establish upper and lower bounds for the variables S_j whereas (3.9) and (3.10) define the x- and the y-variables as binary.

We note that the original model proposed by Correia et al. (2012) includes constraints

$$S_j \geq 0 \quad j \in V \tag{3.11}$$

instead of constraints (3.8). Since the models we are going to present in Chapter 4 limit the start time of each activity $j \in V$ to the time-window $[ES_j, LS_j]$ we decided to consider constraints (3.8) within the continuous-time formulation, above presented.

3.3 Additional concepts and properties

In the MSRCPSP, it is not trivial to check whether two or more activities can be executed in parallel, i.e., can have their execution overlapping for some time. The difficulty emerges from the fact that we have multi-skill resource constraints in addition to the usual precedence ones. This aspect is crucial for the development of algorithms aiming at obtaining feasible solutions for the problem since, as we mentioned above, a small makespan for the project is typically the result of executing activities in parallel. Above all, it is important to have a mechanism, as efficient as possible, to check whether two or more activities can overlap in time. Due to the relevance of this aspect in the general context of the problem at hand, we discuss it here.

3.3.1 Compatibility issues

Suppose that at some time t we have already scheduled some activities. Let us denote by UV the activities still to be scheduled and by W_t a subset of UV containing only activities (not necessarily all) such that all their predecessors are already completed at time t (or that have no predecessors). Can we set to t the start time of all the activities in W_t?

This query has a positive answer if all the resources available at time t can meet the skill requirements of all the activities in W_t at the same time. Such a set of activities with no precedence relations between them and for which there are available resources to process all skills required for their execution is denominated in the literature as a set of compatible activities (cf. Correia et al., 2012). Checking the existence of a precedence relation between each pair of activities can be done straightforwardly. However, as we discuss below, this is not the case when we need to verify whether there are enough resources, among those available at time t, to meet all the skill requirements of the activities in W_t at the same time.

We recall that $\mathcal{Z}_{W_t}$ is the set of resources that are available at time t and that master at least one skill required by at least one activity in W_t. Additionally, we also defined $\mathcal{L}_{W_t}$ as the set of skills required to process the activities in W_t. Checking whether the set of activities W_t is compatible, is something that can be done in polynomial time by solving a flow feasibility problem in an appropriate network that we denote by $G_{W_t} = (V_{W_t}, E_{W_t})$, which is built as follows:

- The set of nodes V_{W_t} contains:
  - a source node v_0 and a sink node v_s;
  - a set of nodes Z_{W_t}, each of which associated with one resource that is available at time t and that masters at least one skill required to process the activities in W_t;
  - a set of nodes $\mathcal{L}_{W_t}$ associated with the skills required to process the activities in W_t.
- The set of arcs E_{W_t} contains:
  - a set of arcs (v_0, k), $k \in Z_{W_t}$ with minimum throughput 0 and capacity 1;
  - a set of arcs (k, l), $k \in Z_{W_t}, l \in (\mathcal{L}^k \cap \mathcal{L}_{W_t})$ with minimum throughput 0 and capacity 1;
  - a set of arcs (l, v_s), $l \in \mathcal{L}_{W_t}$ with minimum throughput and capacity equal to the number of resources necessary to execute skill l for all activities in W_t requiring that skill.

We depict such network in Figure 3.4. If a feasible flow exists in this auxiliary network then we know that there are enough resources to start processing all the activities in W_t, at time t. However, as we

explain next, we can go deeper in this analysis, which, nonetheless, requires some additional concepts. All this information will be useful for the methodological developments that we present (or revisit) in the following sections and chapters.

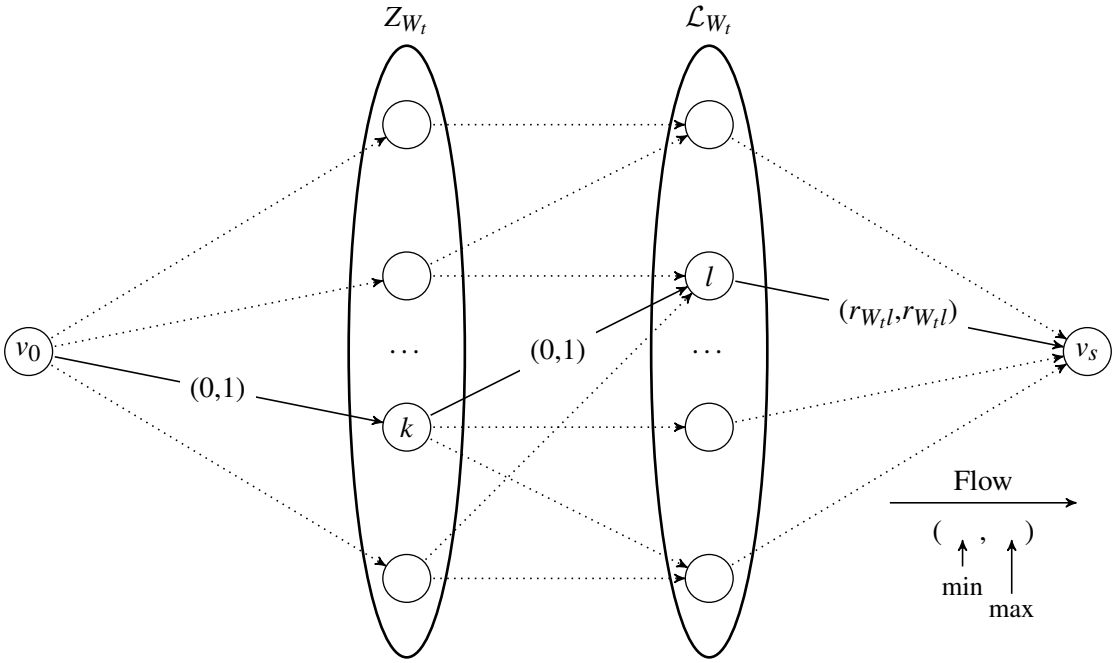

Figure 3.4: A generic G_{W_t}.

3.3.2 Resource weights

Any feasible flow in the above network G_{W_t} induces an assignment of the resources in $\mathcal{Z}_{W_t}$ to the skills required by the activities in W_t. Such a flow may not be unique due to the multi-skill nature of the resources, which may render different possibilities of meeting the skill demands of W_t by varying (i) the selected resources from the set $\mathcal{Z}_{W_t}$ or (ii) the skill $l \in \mathcal{L}_{W_t}$ that each selected resource is assigned to perform, or (iii) both. In fact, each resource k masters a specific set of skills, $\mathcal{L}^k$. Accordingly, by looking into the data, we may characterize a resource as being more versatile than others (e.g., by mastering more skills), more important (e.g., by mastering scarce or highly required skills), etc. Hence, from the point of view of building feasible solutions to our problem, it becomes of great importance to determine the best resource to meet each unitary skill demand, since this assignment may have impact in future iterations and may thus compromise the quality of the derived schedule.

This fact motivated the development of a new concept: the *weight of a resource*. For some resource $k \in \mathcal{R}$, its *weight* is denoted by w_k and represents a measure used for selecting that resource to execute a skill mastered by it and required by at least one activity $j \in W_t$. This new concept aims at providing some insight on the capabilities of the resources to avoid random resource selection and allocation. As seen in the Example 3.1, it is very important to carefully select which resources are assigned to each activity as well as the skill that they will perform.

The advantages of considering this weight measure will be totally clear when we develop the heuristics to this problem in Chapter 5.

3.3.3 Activity priorities

Likewise for the resources, when we look deeply into the activities, we realize that a sort of ranking can be devised. In an attempt to use some rational mechanism for building that rank, a priority value pv_j for

each activity $j \in V \setminus \{0, n+1\}$ can be computed by either using the well-known activity priority rules already proposed for the RCPSP (Kolisch, 1996a, Demeulemeester and Herroelen, 2002) or by initially assigning a random (precedence feasible) priority value to each activity that will be adapted in the course of an algorithm. These assumptions will be discussed in detail in Sections 5.1.1 and 5.2.2, respectively.

3.4 Instances for the MSRCPSP

The following chapters present new models and solution techniques for the problem just described. With the objective of empirically evaluating the performance of such methods, having a comprehensive set of instances of the problem becomes mandatory. We begin this section by identifying and characterizing the parameters that define an instance of the problem in Section 3.4.1. The characteristics of the set of instances generated by Correia et al. (2012) are presented in Section 3.4.2. With the purpose of generating a more comprehensive set of instances, an instance generator was developed in the context of the current thesis. Such generator is presented in Section 3.4.3. A new set of instances was built using it. The characteristics of this new set of instances are described in Section 3.4.4.

3.4.1 Relevant features

Correia et al. (2012) consider three major features that influence the complexity of a MSRCPSP instance: the network complexity (*NC*), the modified resource strength (*MRS*), and the skill factor (*SF*). These parameters are similar to the ones presented in the literature for the RCPSP (see, for instance, Kolisch, 1995 and Kolisch and Sprecher, 1996). The *NC* can be formally defined as the average number of non-redundant arcs in the network, i.e., it is given by the average number of direct successors of each activity in the precedence network (as in the RCPSP). The *SF* measures the proportion of skills required by each activity with regard to the total number of available skills ($SF_j \in]0, 1], j \in V \setminus \{0, n+1\}$). This proportion may be constant for all activities $j \in V \setminus \{0, n+1\}$ or may vary across them. More specifically, we can generate an instance where all the activities require the same proportion of skills and hence have the same value of $SF_j, j \in V \setminus \{0, n+1\}$. It is also possible to generate an instance where a $SF_j, j \in V \setminus \{0, n+1\}$ is randomly selected (within a specific range) for each activity and hence the proportion of skills required may not be the same across all the activities. The *MRS* ($MRS \in]0, \infty[$) measures how demanding the activities are in terms of the available resources. For a given instance, this measure is computed as the ratio between the total number of existing resource units and the total number of resource units needed to perform all the activities.

$$MRS = \frac{K}{\sum_{j \in V} \sum_{l \in \mathcal{L}_j} r_{jl}}$$

Higher values of *MRS* yield instances with more resources or less demanding activities, which, in either case, typically leads to "easier" instances. Therefore, once the total number of activities is fixed as well as a *SF*, and a *MRS*, it is possible to compute the number of resources needed. With the purpose of better illustrating the computation of the *MRS* we present an example extracted from Correia et al. (2012).

Example 3.2 *Consider a project made of 20 activities, each of which requiring 4 skills for being executed; hence a total of 80 skills have to be processed. Let the number of resources required for processing each of those skills be 2 (on average); hence a total of 160 resources are required. If there are 20 resources in the project, the MRS is then equal to 0.1250 (20/160).*

We would like to point out that the skill factor and the modified resource strength described above were adapted (due to the multi-skill nature of the resources) from the resource factor and resource strength, respectively, that are usually considered for the classical RCPSP (cf. Kolisch, 1995).

3.4.2 Existing Instances

In this section, we present the characteristics of the instances built by Correia et al. (2012). Hereafter we denote this set of instances by *Set 1*. Whenever we refer to a parameter as being randomly generated in a set $\{a, \ldots, b\}$, this means that the parameter was generated according to a discrete uniform distribution in that set.

The parameters and their respective values considered for generating these instances are presented below.

- n = 20 activities.
- the processing times of the activities are randomly generated in the set $\{1, \ldots, 10\}$.
- the dummy activity 0 has 3 successors and the dummy activity $n+1$ has 3 predecessors, similarly to the instances of the RCPSP in the PSPLIB—a repository of datasets for various types of resource-constrained project scheduling problems (Kolisch and Sprecher, 1996).
- the number of direct successors of each activity $1, \ldots, n-3$, and the number of direct predecessors of each activity $4, \ldots, n$, were randomly chosen from the set $\{1, 2, 3\}$. It was assured that the desired value of *NC* is met and that the final precedence network is a connected graph.
- 4 skills.
- $NC \in \{1.5, 1.8, 2.1\}$.
- $SF \in \{0.5, 0.75, 1, \text{“variable”}\}$. By “variable”, abbreviated as var., Correia et al. (2012) mean that for each activity, the number of skills it requires was randomly generated in the set $\{2, 3, 4\}$. For instance, $SF = 1$ means that each activity requires the 4 available skills.
- the number of resources varies from 10 to 30 depending on the *SF* and *MRS* of each instance. The *MRS*, the *SF* and the corresponding number of resources are presented in Table 3.6.
- each activity requires $\{1, 2, 3\}$ resources for each skill.
- each resource masters $\{1, 2, 3\}$ skills among the 4 available skills.

Table 3.6: Modified resource strength, skill factor and number of resources for instances in *Set 1*.

$SF = 1$		$SF = 0.75$		$SF = 0.5$		$SF =$ var.	
MRS	*K*	*MRS*	*K*	*MRS*	*K*	*MRS*	*K*
0.1250	20	0.1250	10	0.1250	10	0.1250	10
0.1563	25	0.1667	20	0.1625	13	0.1667	20
0.1875	30	0.2083	25	0.1875	15	0.2083	25

For each combination of *SF*, *NC* and *MRS*, 6 instances were created, resulting in a total of 216 instances.

3.4.3 An instance generator

Despite the efforts undertaken by Correia et al. (2012) to generate a good set of test instances for the MSRCPSP, it is arguable whether that set is representative for this problem. It is worth noticing that defining “a representative set of instances” for some problem is still a matter of debate (cf. Smith-Miles and Bowly, 2015). However, it is relevant to have tools to rationally generate sets of instances of a given problem in order to perform computational tests for evaluating new methodologies. Accordingly, and with the objective of generating a new set of instances that can somehow complement the instances generated by Correia et al. (2012), a comprehensive instance generator is proposed. Furthermore, this generator is built in such a way that it can be easily extended to produce instances for variants and

extensions of the studied problem. For instance, by assuming that each skill corresponds to a pair (skill, level), an instance for a problem considering hierarchical levels of skill is generated.

The proposed methodology for generating instances of the MSRCPSP is somehow inspired by the work developed by Kolisch (1995) for the RCPSP and consists of generating a precedence network, a set of multi-skill resources and a set of activities. The essential algorithms/pseudocodes associated with each phase of the instance generator are presented in this section. The algorithms whose reading can be omitted were included in the Appendix of this thesis.

The proposed instance generator has three components:

1. precedence network generation.
2. multi-skill resource generation.
3. activity generation.

Generation of a Precedence Network

As it was already mentioned above, in a project scheduling problem, the time-dependency between activities can be represented by an AON network $G = (V, E)$ where V is the set of activities and E is the set of arcs. We assume that n is the number of activities to be executed and 0 and $n + 1$ are dummy activities, as discussed before. Each arc in E represents a precedence relation between the two activities it connects. Since the network contains no cycles, we can assume that the nodes (activities) are numbered in such a way that all pairs $(i, j) \in E$ have $i < j$. A precedence network can be obtained by generating precedence relations between pairs of activities. In this process, we attempt to generate a precedence network with no redundancy. An arc (i, j) is said to be redundant if it establishes a precedence that results by transitivity by at least two direct precedences already in the network. For instance, if activity i precedes j and j precedes u, the insertion of the arc (i, u) in the network would introduce redundancy since there is already a transitive precedence, $i \prec\!\!\prec u$ assured by the arcs (i, j) and (j, u).

In order to generate a precedence network, some input data has to be provided. In addition to n and NC, we consider the following.

nStart	:	number of starting activities, i.e., activities having no predecessors.
nFinish	:	number of concluding activities, i.e., activities having no successors.
MaxPred	:	maximum number of predecessors for each activity.
MaxSucc	:	maximum number of successors for each activity.

When generating a precedence network, we must ensure that apart from the initial dummy activity, all the other activities have at least one predecessor. Furthermore, apart from the final dummy activity, all activities must have at least one successor. The previous conditions ensure that the precedence network is connected in the sense that for each $j \in V$, there is at least one path connecting the dummy node 0 with j and at least one path connecting j with the dummy node $n + 1$.

One measure that is often considered when generating precedence networks is the Network Complexity (*NC*) already described.

The procedure for generating a precedence network is summarized in Algorithm 3.1.

We discuss now the four steps itemized in the above-mentioned algorithm.

Step 0 (Initialization).

This step consists of two phases: in the first one we set the first *nStart* nodes (apart from 0) as the starting activities and the last *nFinish* nodes (excluding $n + 1$) as the concluding activities.

In the second phase, arcs are created for connecting the dummy 0 to every starting activity and also for connecting the concluding activities to the dummy $n + 1$ (see Figure 3.5). At the end of this stage, the number of non-redundant arcs in the network is equal to the sum of *nStart* and *nFinish*.

Algorithm 3.1: A procedure for generating a precedence network.

Data: n, NC, $nStart$, $nFinish$, $MaxPred$, $MaxSucc$
Result: A precedence network
begin
  Step 0 (Initialization)
    Set nodes $\{1, \ldots, nStart\}$ to represent the starting activities;
    Set nodes $\{n - nFinish + 1, \ldots, n\}$ to represent the concluding activities;
    Define the arcs $(0, j), j \in \{1, \ldots, nStart\}$;
    Define the arcs $(j, n+1), j \in \{n - nFinish + 1, \ldots, n\}$;
  Step 1
    Assign one predecessor to each activity having no predecessors (apart from activity 0, $n+1$ and the starting activities);
  Step 2
    Assign one successor to each activity having no successors (apart from activity 0, $n+1$ and the concluding activities);
  Step 3
    Add/remove arcs until the desired value for NC is achieved;
end

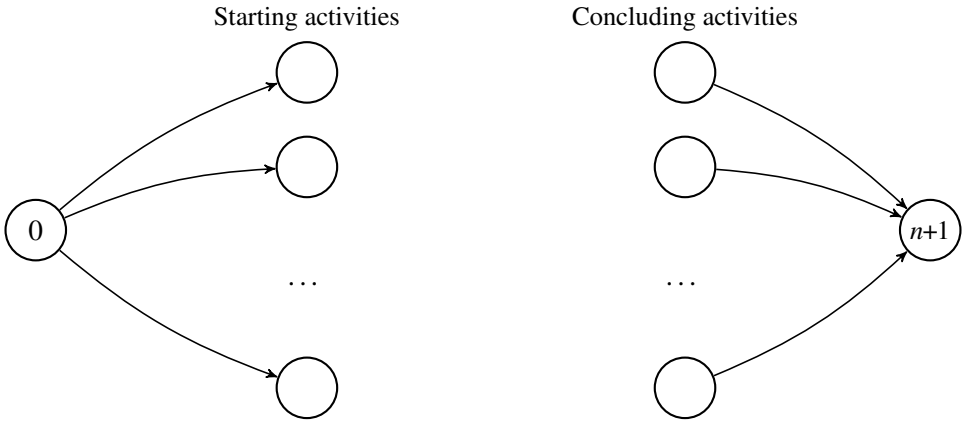

Figure 3.5: Precedence network after the initialization step.

Step 1.

This step is performed for each activity $j \in \{nStart + 1, \ldots, n\}$ (since activities $\{1, \ldots, nStart\}$ have already the dummy 0 as a predecessor) as follows:

(i) randomly[1] select one activity $i \in \{1, \ldots, j-1\}$ such that $\left|Succ(i)\right| < MaxSucc$. Note that if j is a concluding activity, i has to be randomly selected in the set $\{1, \ldots, n - nFinish\}$ to avoid redundancy in the set of concluding activities.

(ii) create the arc (i, j) and increment by one unit the number of non-redundant arcs in the network.

This step is formalized in Algorithm 7.1, presented in the Appendix of this dissertation.

Step 2.

This step excludes activities $\{n - nFinish + 1, \ldots, n\}$ since all these are concluding activities and cannot have other successors than the dummy $n + 1$.

We start with the highest numbered activity (without successors) and proceed backwards. Denote by j one such activity.

[1] all random generated/selected data are pseudo-random numbers that follow a discrete uniform distribution.

We propose the following methodology:

(i) randomly select one activity $u \in \{j+1, \ldots, n\}$ such that $\left|Pred(u)\right| < MaxPred$ and $u \notin Succ(i)$ for $i \in \overline{Pred}(j)$ (i.e., u is not a successor of any predecessor of j). Note that if j is a starting activity, u has to be randomly selected in the set $\{nStart+1, \ldots, n\}$ to avoid redundancy in the set of starting activities.

(ii) define the new arc (j, u) and increment by one unit the number of non-redundant arcs in the network.

We note that in (i) we are avoiding redundancy in the network because if an arc (j, u) is to be considered and if u is already an immediate successor of $i \in \overline{Pred}(j)$, then arc (i, u) would become redundant.

It is also important to point out that in this step we are not following the procedure proposed by Kolisch (1995), which starts assigning successors to the lowest numbered activities. The reason underlying such decision is associated with the fact that using the methodology proposed by Kolisch (1995), a connected precedence network may never be obtained; thus, the procedure may get stuck and requiring to be restarted. This is due to the fact that an upper bound on the maximum number of predecessors per activity, *MaxPred*, is imposed. We illustrate this situation in Example 3.3.

Example 3.3 *Let V = {0,1,2,3,4,5,6}. Thus, we have 5 activities to process (0 and 6 are dummy activities). Assume now that MaxPred = 2, nStart = 1 and nFinish = 1. This means that activity 1 is the only starting activity, activity 5 is the only concluding activity and hence the other activities 2, 3 and 4 must succeed (precede) 1 (5).*

Let activity 5 be an immediate successor of both activities 1 and 2. In this situation, we quickly realize that it is not possible to assign a successor to activity 4. In fact, activity 5 has already reached its maximum number of predecessors $\left|Pred(5)\right| = MaxPred = 2$ *and activity 4, being an intermediate activity, cannot be connected to the dummy node 6.*

Accordingly, the resulting network would not be connected and the process would have to start from the beginning as suggested by Kolisch (1995).

Our proposal for Step 2 fully overcomes this situation. We formalize such step in Algorithm 7.2, presented in the Appendix of this thesis.

Step 3.

This step aims at adding (non-redundant) arcs to the network (thus establishing new precedences) such that the desired value for the *NC* is reached. If the current *NC* is smaller than the desired one we add new arcs into the network by repeating the following steps:

1. randomly select two activities (nodes) i and j with $i < j$ such that $j \notin Succ(i)$ (or $i \notin Pred(j)$).
2. if the new arc (i, j) is not redundant and if the *MaxPred* of activity j and the *MaxSucc* of activity i are not surpassed, then insert this arc into the precedence network and increment the number of non-redundant arcs in the network.

For the selected nodes i and j, $i < j$, Kolisch (1995) identify four types of redundancy that can emerge if the arc (i, j) is created. We identify these types of redundancy in Figure 3.6.

In addition to these types of redundancy we must also consider the ones mentioned in Steps 1 and 2, which are related respectively to having i and j both in $\{n - nFinish + 1, \ldots, n\}$ or both belonging to the set $\{1, \ldots, nStart\}$. In either case, the generated pair is again ignored since no direct precedence can exist between activities inside those sets (otherwise we would have redundancy).

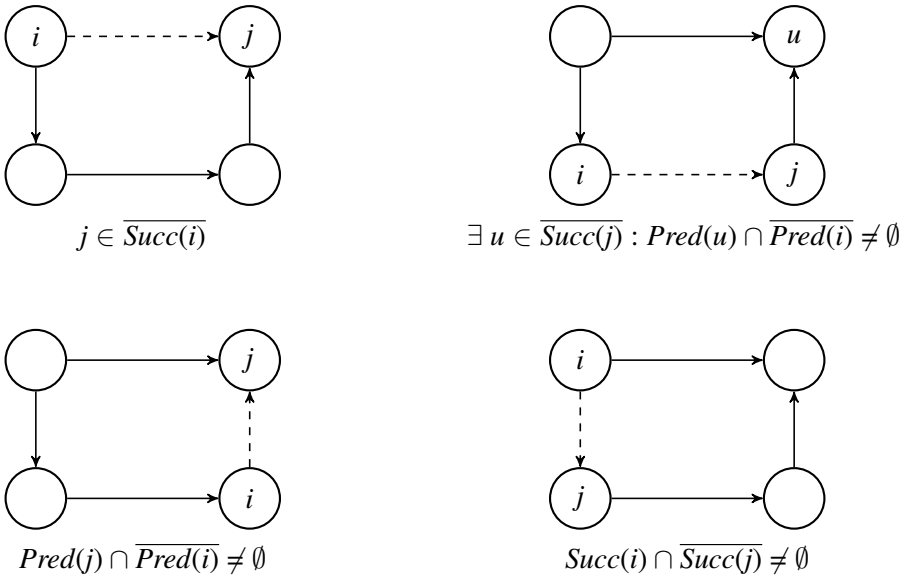

Figure 3.6: Precedence network generation: four types of redundancy identified in Kolisch (1995).

If the existing *NC* is higher than the desired one (this situation may be verified for instance when small values of *NC* are intended) we repeat the following step until we get the desired value:

randomly select an arc $(i,j) \in E$ that can be removed from the network and remove it.

An existing arc (i,j) can be removed from the network as long as the network remains connected. Accordingly, an arc (i,j) can be removed only if node i keeps having at least one successor and node j remains with at least one predecessor.

We note that, again, in this step 3, we are not following exactly the procedure proposed by Kolisch (1995). That procedure (cf. Step 4 of the instance generation algorithm proposed by Kolisch, 1995) only considers adding arcs to the network. In our case, we also consider removing them.

The Step 3 for generating a precedence network is formalized in Algorithm 7.3, presented in the Appendix of this thesis.

Generation of Resources

In order to obtain an instance for the MSRCPSP, we need more than the precedence network generated in the previous section; we also need to generate a set of multi-skill resources. This is the goal of the second phase of our scheme. In addition to $K = \left|R\right|$, this phase also requires:

maxSkills	maximum number of skills a resource can master. Obviously, it should be $1 \leq maxSkills \leq L$.

Each resource is fully defined by the skills it masters. For each resource $k \in \mathcal{R}$ we propose generating the corresponding information as follows:

1. randomly select $\left|\mathcal{L}^k\right|$ number of skills in the set $\{1, \ldots, maxSkills\}$.
2. randomly select $\left|\mathcal{L}^k\right|$ different skills in $\mathcal{L}$, thus obtaining $\mathcal{L}^k$.

After generating the skills mastered by the resources, it is necessary to check if each skill is mastered by at least one resource. If this is not the case, the above procedure is restarted.

Generation of Activities

In the MSRCPSP, each activity $j \in V$ is defined by three components: (i) a processing time (p_j), (ii) a set of required skills ($\mathcal{L}_j$), and (iii) a number of required resources mastering each needed skill (r_{jl}, $l \in \mathcal{L}_j$).

We coordinate the process of generating the second and third components above by making use of two measures already defined previously: the Skill Factor (*SF*) and the Modified Resource Strength (*MRS*).

In this work we consider: (i) instances where all activities have the same skill factor and hence $SF_j = SF, j \in V \setminus \{0, n+1\}$, and (ii) instances where activities can have different skill factors. In this case the number of skills each activity requires is randomly generated. A skill factor of 1 for some activity means that the activity requires at least one unit of every skill.

The *MRS* is an input parameter in the process of generating instances to the MSRCPSP; it makes possible to randomly generate activity skill requirements by calculating the total demand of all activities.

The generation of the information concerning the activities in a MSRCPSP requires the following input, in addition to *SF* and *MRS*:

minProcTime	:	minimum processing time.
maxProcTime	:	maximum processing time.
maxResAct	:	maximum number of resources per activity.
maxResSkill	:	maximum number of resources per each needed skill.

The procedure for generating the information for the activities is summarized in Algorithm 3.2.

Algorithm 3.2: A procedure for generating the information concerning the activities.

Data: *V*, *SF*, *MRS*, *minProcTime*, *maxProcTime*, *maxResAct*, *maxResSkill*
Result: Activities
begin
    Step 1 Processing time generation;
    Step 2 Definition of the sets $\mathcal{L}_j, j \in V \setminus \{0, n+1\}$;
    Step 3 Obtain the desired *MRS*;
end

We discuss now these steps.

Step 1.

This step is accomplished as follows:

1. set $p_0 = p_{n+1} = 0$ (the dummy activities have null processing time).
2. for each activity $j \in \{1, \ldots, n\}$, randomly generate its processing time, p_j, in the set $\{minProcTime, \ldots, maxProcTime\}$.

Step 2.

The definition of the specific skills each activity requires is done as follows:

1. the dummy activities have no skill requirements, i.e., $\left|\mathcal{L}_0\right| = \left|\mathcal{L}_{n+1}\right| = 0$.
2. for each activity $j \in \{1, \ldots, n\}$, we randomly select $\left|\mathcal{L}_j\right|$ different skills in the set $\mathcal{L}$ for defining the set $\mathcal{L}_j$. For a given $l \in \mathcal{L}$, that selection is made by setting the respective r_{jl} to 1. In case all activities require the same number of skills, $\left|\mathcal{L}_j\right|$ is always equal to $\lceil L \times SF \rceil$; otherwise, it should vary.

Since all the available skills are mastered by at least one resource, setting $r_{jl} = 1, l \in \mathcal{L}_j$ is always feasible. Similarly to what we discussed for the resources, it is mandatory that every skill $l \in \mathcal{L}$ is demanded by at least one activity.

We denote by ρ the number of already assigned resources. This value is initialized to 0, and it is incremented as resources are added to the requirements of the activities. At the end of this step, we have $\rho = \sum_{j \in V} |\mathcal{L}_j|$.

This step is formalized in Algorithm 7.5, presented in the Appendix of this thesis.

Step 3.

After selecting the skills required by each activity, we can now focus on meeting the desired *MRS* by incrementing the skill requirements of the activities. We start by computing the total number of resources, which have to be assigned to all activities, in order to meet the *MRS* as follows:

$$\text{total number of resources required by all activities} = \left\lfloor \frac{K}{MRS} \right\rfloor$$

Afterwards, we focus on incrementing the skill requirements of the activities until the sum of their requirements equals the total number of resources required by all activities. It is essential to respect the upper bounds on both the maximum number of resources per activity (*maxResAct*) and the maximum number of resources per each skill required (*maxResSkill*). In every instance generated by this procedure, the minimum number of resources required by each non-dummy activity j is equal to $|\mathcal{L}_j|$, defined in Step 2, i.e., the minimum number of resources per each skill required is assumed to be 1.

Step 3 is illustrated below:

1. while the number of resources already assigned, ρ, is less than the total number of resources required by all activities, randomly select an activity $j \in \{1, \ldots, n\}$ and a random skill $l \in \mathcal{L}_j$, then increment the corresponding value r_{jl} by one unit if $r_{jl} < maxResSkill$ and $\sum_{l' \in \mathcal{L}_j} r_{jl'} < maxResAct$.
2. check whether the skill requirements of activity j can be met. If so, then ρ is incremented by one unit; otherwise r_{jl} is decremented by one unit.

Checking whether the skill requirements of an activity $j \in \{1, \ldots, n\}$ are feasible (in the sense that we have enough resources to perform the activity with those requirements) can be easily done by solving a feasibility problem. In fact, and despite the MSRCPSP being an $\mathcal{NP}$-hard combinatorial optimization problem, as mentioned before, the feasibility of an instance can be checked in polynomial time, as pointed out by Correia et al. (2012). The idea is to check whether a feasible flow exists in the specific network depicted in Figure 3.7 for each activity individually. If it does, we conclude that the generated instance is feasible, since there are enough resources to process one activity at a time and hence it is possible to process all activities, at least, sequentially.

For some activity $j \in \{1, \ldots, n\}$ the corresponding network is built as follows:

- The set of nodes is defined by:
  - a source node v_0;
  - set of nodes $\mathcal{R}_j$ associated with the resources that master at least one skill required by activity j;
  - a set of nodes $\mathcal{L}_j$ associated with the skills required to execute activity j;
  - a sink node v_s.
- The set of arcs contains:
  - a set of arcs (v_0, k), $k \in \mathcal{R}_j$ with minimum throughput 0, capacity 1 and cost 0;

- a set of arcs (k, l), $k \in \mathcal{R}_j, l \in (\mathcal{L}^k \cap \mathcal{L}_j)$ with minimum throughput 0, capacity 1 and cost 0;
- a set of arcs (l, v_s), $l \in \mathcal{L}_j$ with minimum throughput and capacity equal to the current value of r_{jl} and cost 0.

We observe that such network is a particular case of the graph presented in Figure 3.4 when a set W_t consists of only one activity at a time and the available resources are all the existing resources mastering at least one skill required by that activity. We depict this network below, in Figure 3.7.

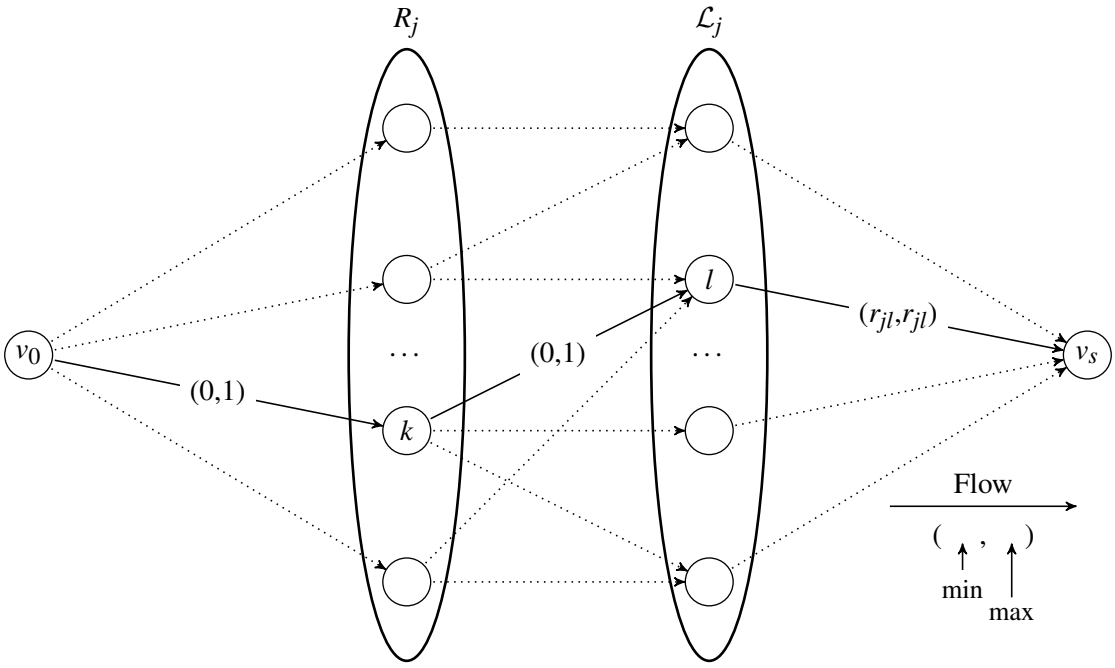

Figure 3.7: Graph for checking if an activity can be processed by the existing resources.

The arcs exiting the source node v_0 and the intermediate arcs have capacity 1 to ensure that each resource is selected at most once and thus can contribute with at most one skill to the execution of the activity. The arcs converging to the sink node v_s have minimum throughput and capacity equal to the total number of resources necessary to perform that skill, r_{jl}, which vary in the course of the algorithm.

If a feasible flow can be found in the referred network, then we know that there are enough resources to meet the requirements of all skills required to process activity j.

Step 3 is formalized in Algorithm 7.6, presented in the Appendix of this thesis.

All the information concerning the proposed instance generator was compiled in the technical report by Almeida et al. (2015).

3.4.4 A new set of instances

A new set of instances for the MSRCPSP was built using the generator just described. Hereafter, this set of instances will be referred to as *Set 2*.

We present below the parameters whose values differ from the ones considered for *Set 1*.

- n = 40 activities.
- each activity requires $\{1, 2, 3, 4, 5, 6, 7\}$ resources for each skill.

- the number of resources varies from 20 to 60 depending on the *SF* and *MRS* of each instance. The *MRS*, the *SF* and the corresponding number of resources are presented in Table 3.10.

Table 3.10: Modified resource strength, skill factor and number of resources for instances in *Set 2*.

SF = 1		*SF* = 0.75		*SF* = 0.5		*SF* = var.	
MRS	*K*	*MRS*	*K*	*MRS*	*K*	*MRS*	*K*
0.0625	40	0.0625	30	0.0625	20	0.0625	30
0.0781	50	0.0792	38	0.0781	25	0.0792	38
0.0938	60	0.0938	45	0.0938	30	0.0938	45

Similarly to *Set 1*, for each combination of SF, NC and MRS, 6 instances were generated, resulting in a total of 216 instances.

Below, we give as an example, the input values used to generate the subset of instances associated with $SF = 0.5$, $NC = 1.5$ and $MRS = 0.0625$ as well as the aspect of one of these instances.

Table 3.11: Instance generator: an example of the input data.

Input	
General Parameters:	*n*: 40
	K: 20
	L: 4
Precedence Network:	*nStart*, *nFinish*: 3
	MaxPred, *MaxSucc*: 3
	NC: 1.5
Resources:	*maxSkills*: 3
Activities:	*MinProcTime*: 1
	MaxProcTime: 10
	SF: 0.5
	MRS: 0.0625
	maxResSkill: 7
	maxResAct: K (no limit on the number resources per activity is imposed)

The output from the instance generator is presented below.

Table 3.12: An instance of the problem: partial precedence network.

j	*Succ*(*j*)
1	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
3	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4	0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
...	
35	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
36	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
37	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
38	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
39	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
40	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The sets $Succ(j), j \in V \setminus \{0, n+1\}$, are used to define the precedence network. For each non-dummy activity *j*, we consider a boolean array with size *n* to represent *Succ*(*j*). Each position of that array is associated with one activity *u*, hence a "1" in that position means that *u* is a direct successor of *j*.

Resources are fully defined by the skills they master. The definition of each activity *j* comprehends

its processing time (p_j) and the number of resources required by each skill needed for their execution: $r_{jl} : l \in \mathcal{L}_j$.

Table 3.13: An instance of the problem: resources.

k	$l = 1$	$l = 2$	$l = 3$	$l = 4$
1	1	0	0	1
2	0	1	1	1
3	0	1	1	1
4	0	1	1	1
5	1	0	1	1
...				
15	0	1	0	0
16	0	0	1	0
17	0	0	0	1
18	1	0	1	1
19	0	1	0	0
20	1	1	1	0

Table 3.14: An instance of the problem: activities.

j	p_j	r_{j1}	r_{j2}	r_{j3}	r_{j4}
1	10	0	3	0	4
2	9	3	3	0	0
3	2	0	0	3	5
4	8	4	0	0	3
5	9	3	0	3	0
...					
35	1	5	0	0	4
36	5	0	4	0	2
37	4	0	5	2	0
38	4	0	0	4	6
39	9	0	5	3	0
40	8	0	5	0	3

3.5 Conclusions

In this chapter, we have thoroughly described the specific multi-skill resource-constrained project scheduling problem being studied in this thesis, which is the problem examined by Correia et al. (2012). The features that characterize the problem were also highlighted.

With the purpose of better depicting some aspects of relevance, an example of the problem was presented and solved, a continuous-time formulation from the literature was revisited and some issues related to determining whether a set of activities can be executed in parallel at a particular moment in time were discussed. We analyzed this last topic in more depth due to its inherent importance in the developments to be presented in the following chapters. Such discussion led us to revisit the well-known activity priority rules and motivated the development of the concept of resource weight with the purpose of devising a sort of ranking for the resources.

To the best of the author's knowledge, no benchmark or well-documented instances for the problem exist in the literature, aside from the ones built by Correia et al. (2012). Moreover, a properly generated set of instances is a mandatory requirement for an adequate evaluation of methodologies. These facts motivated the development of the instance generator formally proposed in this chapter. Using such generator, a set of instances of larger dimensions than the ones referred previously was built. We have discussed the major features that influence the complexity of an instance of the problem and throughly described the characteristics of the two sets of instances considered for the computational experiments to be performed. We noted that the proposed instance generator can be easily adapted to generate instances of extensions of the problem being investigated.

Due to the combinatorial nature of the problem that makes it $\mathcal{NP}$-hard, the methodology proposed by Correia et al. (2012) turned out to be effective only for small-sized instances. In the following chapters, we develop new and possibly more competitive solution methodologies for tackling the specific problem at hand, with a particular emphasis to medium and large-sized instances of it.

CHAPTER 4

Mathematical models and lower bounds

Many mathematical programming formulations can be found in the literature for different project scheduling and management problems. These are usually classified as either discrete or continuous-time formulations, according to the considered time representation. In discrete-time formulations, often referred to as time-indexed formulations, the planning horizon is split into smaller portions, each of which representing a time unit (day, week,..). This type of mathematical models is known for usually requiring a large number of binary variables, each of which being associated with some event that may occur at a specific moment in time. The events are usually related to the status of the activities: start, finish, or in progress. The number of binary variables involved in this type of models renders heavy mathematical programming formulations that easily become intractable to solve medium or large-sized instances to optimality using an optimization solver. Nevertheless, these formulations are known to yield sharp linear programming relaxation bounds.

Another kind of formulations that have been proposed for project scheduling problems are the so-called continuous-time formulations. These mathematical formulations consider continuous variables to represent the start times of the activities involved in the project. These formulations require less variables than the time-indexed models. However, typically the optimal values of their corresponding linear programming relaxations are poor.

Next, we are going to give an overview of some mathematical models that have already been proposed for the RCPSP. The basic discrete-time formulation was proposed by Pritsker et al. (1969) who study a multi-project RCPSP. Later, Christofides et al. (1987) proposed a discrete-time mathematical model featuring disaggregated precedence relations. This formulation, referred to in the literature as DDT—see for instance Schwindt and Zimmermann (2015), short for disaggregated discrete-time formulation, is among those yielding the strongest linear programming relaxation bounds. It differs from the one of Pritsker et al. (1969) only in how the precedence constraints are formulated. Mingozzi et al. (1998) proposed a discrete-time formulation where it is assumed that all feasible sets of activities are known. A feasible set of activities contains all the activities that may be processed simultaneously without violating precedence or resource constraints. One of the disadvantages of this formulation is associated with the number of feasible sets of activities, which can be exponential. An opposite approach was followed by Alvarez-Valdés and Tamarit (1993), who presented a continuous-time formulation for the RCPSP alongside with a set of valid inequalities assuming that all minimal forbidden sets of activities are known. A forbidden set of activities comprehends the activities that despite not being linked by any kind of prece-

dence relations, cannot be processed in parallel due to resource limitations. A forbidden set is said to be minimal, when none of its subsets is a forbidden set. The main objective is to introduce additional disjunctive precedence relations in the precedence network. For comprehensive reviews on mathematical formulations for resource-constrained project scheduling problems the reader should refer to the work of Demassey (2008), Koné et al. (2011), and Artigues et al. (2015). More recently, in a paper by Artigues (2017) the strength of some discrete-time formulations for the RCPSP is deeply analyzed.

Although large-scale instances of the MSRCPSP are expected to be tackled only by approximate procedures, the development of adequate formulations is of great relevance since, often, they are a means for obtaining lower bounds that, in turn, are crucial for evaluating the quality of heuristic solutions. Moreover, the use of an off-the-shelf solver for tackling a model is often the only tool available to practitioners who do not master advanced optimization and computational techniques. Hence, the choice of an adequate model may render a successful resolution of a problem.

The remainder of this chapter is organized as follows. In Section 4.1, we revisit the reduction test and other inequalities proposed by Correia et al. (2012) for the continuous-time formulation developed therein. In Section 4.2, which is devoted to the study of discrete-time formulations, we revisit the model of Montoya et al. (2014) with the corrections made by Correia and Saldanha-da-Gama (2015b), and we propose two new mathematical formulations. Section 4.3 presents two new methods for computing lower bounds for the problem being studied. This chapter ends with some concluding remarks in Section 4.4.

A good upper bound on the makespan of the project is required to both load some valid inequalities and tighten the start time-windows of the activities (particularly for the discrete-time formulations). The methods for computing such upper bounds are only presented in Chapter 5. Hence, we opted to report in Section 5.3.3, the results associated with the developments proposed in this chapter, after the heuristics for the problem have been formally described and their respective results presented.

4.1 Enhancements for a continuous-time model

We recall that Correia et al. (2012) proposed the MILP formulation already revisited in page 23 for the particular problem being investigated. Aside from that mathematical formulation, such paper also introduces a reduction test and other additional inequalities with the purpose of fixing some variables at their optimal values, which may contribute to increase the number of instances successfully solved (cf. Correia et al., 2012). Since these additional developments were considered for the numerical experiments to be performed and with the objective of keeping this thesis self-contained, we opted to include them in this section. We begin by defining the concept of pairs of incompatible activities (in opposition to the notion of pairs of compatible activities introduced in Section 3.3.1). Then, we revisit the developments proposed by Correia et al. (2012), namely the derived reduction test, in Section 4.1.1, and the other inequalities proposed, in Section 4.1.2.

The definition of pairs of incompatible activities is mandatory for the proper comprehension of the developments presented in the following sections, therefore we present it here.

Definition 4.1 *(cf. Correia et al., 2012) Two activities $i, j \in V, i, j \notin \{0, n{+}1\}$ are said to be incompatible if there is never a slot in time in which they can be executed simultaneously. Activities i and j are said to be compatible (see Section 3.3.1) if they are not incompatible.*

The previous incompatibility definition has already been proposed for the RCPSP (cf. Alvarez-Valdés and Tamarit, 1993 or Klein and Scholl, 1999) but their extension to the MSRCPSP entails an increased complexity associated with verifying whether there are enough multi-skill resources to process those two activities simultaneously.

Consider two activities $i, j \in V$. If at least one of the following conditions hold, then these activities are incompatible.

1. There is a precedence relation between i and j;
2. $LS_i + p_i \leq ES_j$;
3. $LS_j + p_j \leq ES_i$;
4. No feasible flow exists in the auxiliary network $G_{W_t} = (V_{W_t}, E_{W_t})$ depicted in Figure 3.4. To build such network, the set W_t is considered to be made by exclusively activities i and j (refer to Section 3.3.1 for a detailed description regarding the construction of G_{W_t}).

4.1.1 Reduction tests

After determining all pairs of incompatible activities, it is possible to set the values of some variables in the model P_{CT}. Correia et al. (2012) developed a reduction test with the objective of fixing the values of some variables y_{ij} $(i, j \in V)$ at their optimal values. In this section, we present such reduction test.

Consider two activities i and j and assume that they are compatible, i.e., that the value of y_{ij} can still be chosen. Assume additionally that activity i cannot finish before j. Given these assumptions, we know that y_{ij} cannot have the value of 1 and hence we can set it to 0. If we consider that the same two activities are now incompatible with no precedence relation existing between them, we know that activity i must be executed before j or j must be processed before i, i.e., that the value of y_{ij} can still be chosen.

In some situations, it is possible to determine this processing order beforehand. In that case, we may realize that activity i has to be processed before activity j and in such situation it is clear that all successors of the j become successors of i, by transitivity.

We now formalize the reduction test proposed by Correia et al. (2012) in Algorithm 4.1. We note that lines 8–20 of Algorithm 4.1 consist of an adaptation to the MSRCPSP of the well-known *pair test* proposed by Carlier and Pinson (1989) for the job-shop scheduling problem.

Algorithm 4.1: Reduction test proposed by Correia et al. (2012).

Data: V; $Pred(j), j \in V$; $Succ(j), j \in V$; $r_{jl}, j \in V, l \in \mathcal{L}_j$
Result: Fixing some variables y_{ij} at their optimal values

```
1  begin
2      Consider two activities i and j;
3      if i and j are compatible then
4          if ES_i + p_i > LS_j then
5              y_ij = 0;
6          end
7      end
8      if i and j are incompatible with no precedence relation between them then
9          if ES_i + p_i > LS_j then
10             y_ji = 1, y_ij = 0;
11             y_js = 1, y_sj = 0, for all successors s of i;
12             y_pi = 1, y_ip = 0, for all predecessors p of j;
13             y_ps = 1, y_sp = 0, for all predecessors p of j and for all successors s of i;
14         else if ES_j + p_j > LS_i then
15             y_ij = 1, y_ji = 0;
16             y_is = 1, y_si = 0, for all successors s of j;
17             y_pj = 1, y_jp = 0, for all predecessors p of i;
18             y_ps = 1, y_sp = 0, for all predecessors p of i and for all successors s of j;
19         end
20     end
21 end
```

4.1.2 Other inequalities

Correia et al. (2012) derived several sets of valid inequalities by investigating the relationship between incompatible activities and the selection of disjunction variables. In this section, we revisit such valid inequalities whose development is strongly related to the well-known notion of forbidden sets widely used in the context of the RCPSP.

The information concerning the sets of incompatible activities can be easily introduced into model P_{CT}. As discussed previously, when two activities i and j are incompatible, either i is executed before j or the other way around. Accordingly, if the constraint $y_{ij} + y_{ji} \leq 1$ exists in the model then it can be replaced by

$$y_{ij} + y_{ji} = 1, \tag{4.1}$$

since in this case we know beforehand that it is not possible to have $y_{ij} = y_{ji} = 0$.

Any pair of compatible activities $\{i, j\}$ can be extended to a triplet of incompatible activities $\{i, j, u\}$ if both $\{i, u\}$ and $\{j, u\}$ are pairs of compatible activities and the three activities i, j and u cannot be processed simultaneously. Checking if three activities can be executed simultaneously reduces to determining whether a feasible flow exists in an auxiliary directed network $G_{W_t} = (V_{W_t}, E_{W_t})$ built as before, but now including the skill requirements of the three activities and the resources that may perform them.

The information regarding these triplets of incompatible activities can also be used to introduce new inequalities into model P_{CT}. For each triplet of incompatible activities $\{i, j, u\}$, we add the following valid inequality to P_{CT}:

$$y_{ij} + y_{ji} + y_{iu} + y_{ui} + y_{ju} + y_{uj} \geq 1. \tag{4.2}$$

In fact, if the three activities i, j, u are incompatible, then at least one of the disjunction variables involved in the inequality presented above has to be equal to 1, i.e., at most two of those activities may be in progress at the same time.

Naturally, the logic followed to derive inequality (4.2) could be used to obtain inequalities for sets of more than three incompatible activities. Obviously, a trade-off between the benefits of introducing such inequalities into the model and the computational effort required to derive them must be reached. Correia et al. (2012) did not determined the sets of four incompatible activities such that all its triplets are compatible due to the associated increase in the computational effort.

Correia et al. (2012) investigated other inequalities already existing for project scheduling problems, and presented the following two results whose proofs are straightforward. We recall that the y_{ij} $(i, j \in V)$ variables are only necessary for pairs of activities having no precedence relations between them.

Result 4.1 *(cf. Correia et al., 2012) Let $u, i, j \in V$ be such that no precedence relation exists between them.*

If

u and j are not incompatible

or

If

i) u and j are incompatible and
ii) u and i are incompatible and
iii) i and j are incompatible

Then it must be

$$y_{uj} \geq y_{ui} + y_{ij} - 1. \tag{4.3}$$

Result 4.2 *(cf. Correia et al., 2012) Let $u, i, j \in V$ be such that no precedence relation exists between them.*

1. *If*

   i) u precedes i, according to the reduction test presented and

   ii) u and j are not incompatible

   or

   If

   i) u precedes i, according to the reduction test presented and

   ii) u and j are incompatible and

   iii) i and j are incompatible

   Then it must be

   $$y_{uj} \geq y_{ij}. \tag{4.4}$$

2. *If*

   i) i precedes j, according to the reduction test presented and

   ii) u and j are not incompatible

   or

   If

   i) i precedes j, according to the reduction test presented and

   ii) u and j are incompatible and

   iii) i and u are incompatible

   Then it must be

   $$y_{uj} \geq y_{ui}. \tag{4.5}$$

In addition to the previous sets of inequalities, it is possible to derive other valid inequalities for the MSRCPSP by, for instance, making use of the binary variables associated with the assignment of the resources. In fact, when we have an upper bound UB on the optimal makespan of the project (i.e., the optimal value of the problem) we conclude immediately that, in an optimal solution to the problem, no resource $k \in \mathcal{R}$ spends, in total, more time than UB for processing the activities that it is assigned to perform. Hence, we can write:

$$\sum_{j \in V_k} \sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} p_j x_{jlk} \leq UB, \qquad k \in \mathcal{R}. \tag{4.6}$$

Considering that in the previous constraints we have binary variables in the left term, we can divide both members of 4.6 by an integer value $q \in \{1, \ldots, \max_{j \in V_k}\{p_j\}\}$, obtaining:

$$\sum_{j \in V_k} \sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} \frac{p_j}{q} x_{jlk} \leq \frac{UB}{q}, \qquad k \in \mathcal{R},\ q \in \{1, \ldots, \max_{j \in V_k}\{p_j\}\}.$$

By rounding down each coefficient in the left-hand side of the constraints just presented and then by performing the same to the right-hand side, we obtain a new set of additional inequalities that can be added to model P_{CT}:

$$\sum_{j \in V_k} \sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} \left\lfloor \frac{p_j}{q} \right\rfloor x_{jlk} \leq \left\lfloor \frac{UB}{q} \right\rfloor, \qquad k \in \mathcal{R},\ q \in \{1, \ldots, \max_{j \in V_k}\{p_j\}\}. \tag{4.7}$$

A deeper analysis reveals that some of the inequalities (4.7) may have the same right hand-side, and in that case there may be inequalities that dominate others. Hence, for each distinct right-hand side, we only need to add to model P_{CT} the inequality with the "strongest" left-hand side.

4.2 Discrete-time models

Discrete-time, often referred to as time-indexed formulations have been widely applied to the RCPSP and its variants. Therefore, it is natural to develop such kind of formulations also to the MSRCPSP. The formulations we are presenting next include time-indexed binary variables that are equal to one for a particular time t if the corresponding activity starts at that time and zero otherwise. To consider such variables, we assume a planning horizon $\mathcal{T} = \{0, \ldots, UB\}$ where UB is an upper bound previously found for the makespan. In addition to those binary variables, these formulations also consider time-indexed variables associated with the resources. In this type of formulations, the number of variables and constraints can be huge. In fact, they depend on length of the planning horizon that, in turn, strongly depends on several factors such as the magnitude of the processing times of the activities and their skill requirements, the network complexity of the precedence network and the number of resources that master each skill.

This section is organized as follows. In Section 4.2.1, we revisit a mathematical model from the literature. In Section 4.2.2, we develop two new discrete-time formulations. We conclude this part by presenting some additional inequalities for the discrete-time formulations based on the work developed by Correia et al. (2012), presented in the previous section.

4.2.1 A model from the literature

We start this section by revisiting the discrete-time formulation by Montoya et al. (2014) with the changes proposed by Correia and Saldanha-da-Gama (2015b). We denote this model by P_M. It requires the following sets of decision variables:

$$S_{jt} = \begin{cases} 1, & \text{if activity } j \text{ starts at time instant } t; \\ 0, & \text{otherwise.} \end{cases} \qquad j \in V, t \in \{ES_j, \ldots, LS_j\}$$

$$x_{jkt} = \begin{cases} 1, & \text{if resource } k \text{ starts processing activity } j \text{ at time } t; \\ 0, & \text{otherwise.} \end{cases} \qquad k \in \mathcal{R}, j \in V_k, t \in \{ES_j, \ldots, LS_j\}$$

$$y_{jlk} = \begin{cases} 1, & \text{if resource } k \text{ is assigned to perform skill } l \text{ for activity } j; \\ 0, & \text{otherwise.} \end{cases} \qquad k \in \mathcal{R}, j \in V_k, l \in \mathcal{L}^k \cap \mathcal{L}_j$$

The formulation of P_M is as follows.

$$\min \quad \sum_{t=ES_{n+1}}^{LS_{n+1}} t\, S_{n+1,t} \tag{4.8}$$

subject to:

$$\sum_{t=ES_j}^{LS_j} t\, S_{jt} \geq \sum_{t=ES_i}^{LS_i} t\, S_{it} + p_i \qquad i,j \in V \wedge (i,j) \in E \tag{4.9}$$

$$\sum_{t=ES_j}^{LS_j} x_{jkt} \leq 1 \qquad k \in \mathcal{R}, j \in V_k \tag{4.10}$$

$$\sum_{j \in V_k} \sum_{\tau=\max\{ES_j, t-p_j+1\}}^{\min\{LS_j,t\}} x_{jk\tau} \leq 1 \qquad k \in \mathcal{R}, t \in \mathcal{T} \tag{4.11}$$

$$x_{jkt} \leq S_{jt} \qquad j \in V \setminus \{0, n+1\}, k \in \mathcal{R}_j, t \in \{ES_j, \ldots, LS_j\} \tag{4.12}$$

$$x_{jkt} + 1 \geq S_{jt} + \sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} y_{jlk} \qquad j \in V \setminus \{0, n+1\}, k \in \mathcal{R}_j, t \in \{ES_j, \ldots, LS_j\} \tag{4.13}$$

$$\sum_{k \in \mathcal{R}^l} y_{jlk} = r_{jl} \qquad j \in V \setminus \{0, n+1\}, l \in \mathcal{L}_j \tag{4.14}$$

$$\sum_{t=ES_j}^{LS_j} x_{jkt} = \sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} y_{jlk} \qquad j \in V \setminus \{0, n+1\}, k \in \mathcal{R}_j \tag{4.15}$$

$$S_{jt} \in \{0,1\} \qquad j \in V, t \in \{ES_j, \ldots, LS_j\} \tag{4.16}$$

$$x_{jkt} \in \{0,1\} \qquad k \in \mathcal{R}, j \in V_k, t \in \{ES_j, \ldots, LS_j\} \tag{4.17}$$

$$y_{jlk} \in \{0,1\} \qquad k \in \mathcal{R}, j \in V_k, l \in \mathcal{L}^k \cap \mathcal{L}_j \tag{4.18}$$

The objective function (4.8) represents the makespan (to be minimized). Constraints (4.9) assure that the precedence relations between each pair of activities (i,j) belonging to the precedence network hold. Constraints (4.10) guarantee that a resource can only start performing an activity within its start time-window. Constraints (4.11) restrict the allocation of a resource to at most one activity at each time instant. Constraints (4.12) and (4.13) ensure the consistency of the start times of the activities with their assigned resources. More precisely, from constraints (4.12), if resource k starts performing activity j at time t ($x_{jkt} = 1$) then activity j must start at time t ($S_{jt} = 1$). The presence of constraints (4.13) together with the sets of constraints (4.10), (4.14) and (4.15) prevent an activity j' from being associated with two start times, i.e., it is not possible to have $S_{j't_1} = 1$ and $S_{j't_2} = 1$ for $t_1 \neq t_2$. Therefore, the widely used constraints $\sum_{t=ES_j}^{LS_j} S_{jt} = 1$, $j \in V \setminus \{0, n+1\}$ (e.g., Pritsker et al., 1969 for the RCPSP and the two formulations proposed next) are implicitly assured in this model. Constraints (4.14) assure that the skill requirements of each activity are met through the assignment of the necessary resources to each skill needed for its execution. Constraints (4.15) restrict the contribution of a resource to one skill for each activity it is assigned to perform. Constraints (4.16)–(4.18) define the variables as being binary.

Christofides et al. (1987) proposed the well-known disaggregated discrete-time (*DDT*) formulation for the RCPSP whose linear programming relaxation renders better lower bounds than the *DT* model proposed by Pritsker et al. (1969). We recall that the difference between models *DT* and *DDT* concerns the way in which precedence constraints are modeled—the latter considers a disaggregated version of the precedence constraints that appear in the former.

Using a similar reasoning to the one adopted by Christofides et al. (1987), we can derive an alternative model for the MSRCPSP—P_{MDDT}. To that end, we have to replace constraints (4.9) in model P_M with the following disaggregated set of precedence constraints:

$$\sum_{\tau=t}^{LS_i} S_{i\tau} + \sum_{\tau=ES_j}^{\min\{LS_j,t+p_i-1\}} S_{j\tau} \leq 1, \quad (i,j) \in E, t \in \{ES_i, \ldots, LS_i\} \tag{4.19}$$

Constraints (4.19) state that if activity i is a predecessor of activity j then the execution of activities i and j cannot overlap in any time t.

However, if we solve the model immediately after performing this replacement, the resulting optimal value, for any instance of the problem, is always zero since all the variables $S_{n+1\,t}$, $t \in \{ES_{n+1}, ..., LS_{n+1}\}$ are zero. This is explained by the fact that there is no constraint in model P_M forcing the execution of the dummy activity $n + 1$ other than the standard precedence constraints (4.9), which were replaced by constraints (4.19). In fact, the remaining constraints featuring the start time variables of the activities are used to ensure that the skill requirements of the activities are met through the assignment of the necessary resources, and hence they are not applicable to activity $n + 1$, which has null skill requirements. For all pairs of activities consisting of an activity with no non-dummy successors (i.e., whose only successor is activity $n + 1$) and the activity $n + 1$, the constraints (4.19) only imply that the former activity cannot be executed at the same time as the dummy activity $n + 1$, and hence they do not assure that activity $n + 1$ is processed. Thus, we added the set of constraints $\sum_{t=ES_j}^{LS_j} S_{jt} = 1$, $j \in V$ into the model, obtaining P_{MDDT}. We note that it would only be mandatory to add such constraint for activity $n + 1$ but since all the optimization models to be presented in the following sections (and considered in the computational experiments to be performed) include this constraint for all activities, we decided to supply that same information to model P_{MDDT}.

4.2.2 New models

In this section, we present two new discrete-time formulations for the MSRCPSP.

Model P_{DT}

We start by introducing a new formulation for the MSRCPSP that results from adapting to our problem the well-known discrete-time formulation for the RCPSP proposed by Pritsker et al. (1969).

This new model makes use of the following variables:

$$S_{jt} = \begin{cases} 1, & \text{if activity } j \text{ starts at time instant } t; \\ 0, & \text{otherwise.} \end{cases} \qquad j \in V, t \in \mathcal{T}$$

$$z_{jlkt} = \begin{cases} 1, & \text{if resource } k \text{ starts performing skill } l \text{ for activity } j \text{ at time instant } t; \\ 0, & \text{otherwise.} \end{cases} \qquad k \in \mathcal{R}, j \in V_k, l \in \mathcal{L}^k \cap \mathcal{L}_j, t \in \mathcal{T}$$

Considering the previous decision variables, model P_{DT} establishes a new time-indexed formulation to the MSRCPSP. We define model P_{DT} as follows.

$$\min \sum_{t=ES_{n+1}}^{LS_{n+1}} t\, S_{n+1,t} \tag{4.20}$$

subject to:

$$\sum_{t=ES_j}^{LS_j} S_{jt} = 1 \qquad j \in V \tag{4.21}$$

$$\sum_{t=ES_j}^{LS_j} t\, S_{jt} \geq \sum_{t=ES_i}^{LS_i} t\, S_{it} + p_i \qquad i,j \in V \wedge (i,j) \in E \tag{4.22}$$

$$\sum_{t=ES_j}^{LS_j} \sum_{k \in \mathcal{R}^l} z_{jlkt} = r_{jl} \qquad j \in V \setminus \{0, n+1\}, l \in \mathcal{L}_j \tag{4.23}$$

$$\sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} z_{jlkt} \leq S_{jt} \qquad k \in \mathcal{R}, j \in V_k, t \in \{ES_j, \ldots, LS_j\} \tag{4.24}$$

$$\sum_{j \in V_k} \sum_{\tau=\max\{ES_j, t-p_j+1\}}^{\min\{LS_j,t\}} \sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} z_{jlk\tau} \leq 1 \qquad k \in \mathcal{R}, t \in \mathcal{T} \tag{4.25}$$

$$S_{jt} \in \{0,1\} \qquad j \in V, t \in \{ES_j, \ldots, LS_j\} \tag{4.26}$$

$$z_{jlkt} \in \{0,1\} \qquad k \in \mathcal{R}, j \in V_k, l \in \mathcal{L}^k \cap \mathcal{L}_j, \; t \in \{ES_j, \ldots, LS_j\} \tag{4.27}$$

The objective function (4.20) represents the start time of the final dummy activity, which is equivalent to the completion time of the entire project (that is to be minimized). Constraints (4.21) state that every activity starts exactly once. Constraints (4.22) ensure that precedence relations hold for every pair of activities $(i,j) \in E$. Constraints (4.23) guarantee that the skill requirements of all activities are fulfilled through the assignment of the adequate resources. Constraints (4.24) impose that if a resource is performing a skill l for an activity j it has to start performing that skill at the start time of j. Constraints (4.25) assure that each resource is not assigned to more than one activity at a time. Constraints (4.26) and (4.27) define the domain of the decision variables.

Unlike P_{MDDT}, model P_{DDT} is obtained straightforwardly by replacing only the precedence constraints (4.22) by the disaggregated precedence relations (4.19).

Model P_N

A new discrete-time model for the MSRCPSP can be obtained by combining the structures of the formulations P_M and P_{DT}. Let us denote by P_N this new model. This new formulation uses the variables S_{jt}, x_{jkt}, and y_{jlk} already used in model P_M, and is defined as follows.

$$\min \sum_{t=ES_{n+1}}^{LS_{n+1}} t\, S_{n+1,t} \tag{4.28}$$

subject to:

$$\sum_{t=ES_j}^{LS_j} S_{jt} = 1 \qquad j \in V \tag{4.29}$$

$$\sum_{t=ES_j}^{LS_j} t\, S_{jt} \geq \sum_{t=ES_i}^{LS_i} t\, S_{it} + p_i \qquad i,j \in V \wedge (i,j) \in E \tag{4.30}$$

$$x_{jkt} \leq S_{jt} \qquad j \in V \setminus \{0, n+1\}, k \in \mathcal{R}_j, t \in \{ES_j, \ldots, LS_j\} \tag{4.31}$$

$$\sum_{t=ES_j}^{LS_j} x_{jkt} = \sum_{l \in \mathcal{L}^k \cap \mathcal{L}_j} y_{jlk} \qquad j \in V \setminus \{0, n+1\}, k \in \mathcal{R}_j \tag{4.32}$$

$$\sum_{k \in \mathcal{R}^l} y_{jlk} = r_{jl} \qquad j \in V \setminus \{0, n+1\}, l \in \mathcal{L}_j \tag{4.33}$$

$$\sum_{j \in V_k} \sum_{\tau=\max\{ES_j, t-p_j+1\}}^{\min\{LS_j, t\}} x_{jk\tau} \leq 1 \qquad k \in \mathcal{R}, t \in \mathcal{T} \tag{4.34}$$

$$S_{jt} \in \{0,1\} \qquad j \in V, t \in \{ES_j, \ldots, LS_j\} \tag{4.35}$$

$$x_{jkt} \in \{0,1\} \qquad k \in \mathcal{R}, j \in V_k, t \in \{ES_j, \ldots, LS_j\} \tag{4.36}$$

$$y_{jlk} \in \{0,1\} \qquad k \in \mathcal{R}, j \in V_k, l \in \mathcal{L}^k \cap \mathcal{L}_j \tag{4.37}$$

The objective function (4.28) accounts for the start time of the dummy n + 1 (to be minimized), i.e, the makespan. Constraints (4.29) assure that every activity starts exactly once. Constraints (4.30) express the precedence relations of every pair of activities $(i,j) \in E$. Constraints (4.31) ensure that if a resource $k \in \mathcal{R}_j$ starts executing activity j at time t, then activity j starts its execution at time t. Constraints (4.32) impose that a resource $k \in \mathcal{R}_j$ contributes with one skill $l \in \mathcal{L}_j \cap \mathcal{L}^k$ to activity j if it has been assigned to perform that activity. Constraints (4.33) ensure that the skill requirements of all activities are fulfilled. Constraints (4.34) ensure that a resource is not assigned to more than one activity at a time. Constraints (4.35)–(4.37) are domain constraints.

Likewise to the substitution performed to P_{DT} in order to obtain P_{DDT}, P_{NDDT} is obtained by replacing the constraints (4.30) in model P_N by the disaggregated precedence constraints (4.19).

4.2.3 Valid inequalities for the discrete-time models

Similarly to what was observed for the RCPSP, we also expect that the substitution of the classical precedence constraints by the disaggregated ones (inequalities 4.19) in the discrete-time models presented in the previous sections will contribute to the enhancement of the lower bound values stemming from their corresponding linear programming relaxations.

Still with the objective of both strengthening the linear programming relaxation of such models and improving their performance in integer solving, we discretized the set of constraints (4.7) introduced by Correia et al. (2012) for the continuous-time formulation developed therein (cf. Section 4.1.2).

For the mathematical model P_{DT} such discretization is straightforward.

$$\sum_{j \in \mathcal{V}_k} \sum_{l \in \mathcal{L}_j \cap \mathcal{L}^k} \sum_{t=ES_j}^{LS_j} \left\lfloor \frac{p_j}{q} \right\rfloor z_{jlkt} \leq \left\lfloor \frac{UB}{q} \right\rfloor, \quad k \in \mathcal{R}, q \in \{1, \ldots, \max_{j \in V_k}\{p_j\}\} \tag{4.38}$$

Due to the type of variables involved in models P_M and P_N, two sets of valid inequalities are derived for these formulations.

$$\sum_{j \in \mathcal{V}_k} \sum_{l \in \mathcal{L}_j \cap \mathcal{L}^k} \left\lfloor \frac{p_j}{q} \right\rfloor y_{jlk} \leq \left\lfloor \frac{UB}{q} \right\rfloor, \quad k \in \mathcal{R}, q \in \{1, \ldots, \max_{j \in V_k}\{p_j\}\} \tag{4.39}$$

$$\sum_{j \in \mathcal{V}_k} \sum_{t=ES_j}^{LS_j} \left\lfloor \frac{p_j}{q} \right\rfloor x_{jkt} \leq \left\lfloor \frac{UB}{q} \right\rfloor, \quad k \in \mathcal{R}, q \in \{1, \ldots, \max_{j \in V_k}\{p_j\}\} \tag{4.40}$$

4.3 Lower bounds

The most obvious lower bound for project scheduling problems with scarce resources is obtained by discarding the resource constraints. The resulting problem corresponds to finding the longest path between the dummy nodes 0 and $n + 1$ in the precedence network—the critical path. The length of the critical path is known to be a poor lower bound when multi-skill resources are involved, as discussed previously. Typically, in such situation we have to rely on other methods for computing better lower bounds. Throughout this section, it is assumed that we are dealing with an optimization problem with a minimization objective function, such as the particular project scheduling problem being studied.

A lower bound D on the optimal value of a specific instance of the MSRCPSP indicates that the project lasts at least D time units in the optimal solution of the referred instance. The higher the value of a lower bound, the better its quality. In the absence of optimal values, lower bounds become particularly useful to evaluate the performance of approximate methods and to be incorporated into both optimization models, such as the ones presented previously, and exact methods, in an attempt to reduce the search space. The quality of a feasible solution derived by a heuristic is generally evaluated by computing a so-called *gap* between the associated objective value and a lower bound. When good lower and upper bounds are used in a branch-and-bound framework, the number of nodes in the tree may be substantially reduced.

The development of good techniques for computing lower bounds for complex optimization problems is hence of great importance, namely when dealing with instances of large dimensions whose optimal values are often unachievable even when a high limit on the computational time is imposed (cf. Tables 5.8 and 5.10).

We observe that a lot of effort has been put into developing this type of algorithms for the class of resource-constrained project scheduling problems. Lower bound computing methods are usually classified as constructive or destructive. A constructive procedure solves a relaxed version of the original problem. On the other hand, a destructive method, also referred to as destructive improvement techniques (cf. Klein, 2000) is based upon setting a value D on the objective function of the problem (usually provided by a constructive lower bound method) and in incrementing such value until no infeasibilities are detected (i.e., it cannot be proved that the objective function value D is not associated with a feasible solution). The largest value of D that corresponds to a solution whose infeasibility can be proved

is then the best lower bound. Binary search can also be applied within destructive methods, considering additionally a value for the upper bound, which can be provided, for instance, by a heuristic. For a comprehensive description and review of the work done in terms of the development of lower bound procedures for the RCPSP, the reader should refer to Klein (2000) and to Schwindt and Zimmermann (2015).

From the literature reviewed in Chapter 2, we observe that only a modest amount of effort has been put into developing lower bound procedures for project scheduling problems with multi-skilled resources. The mechanisms proposed consist of destructive improvement techniques adapted from the RCPSP.

In this section, we present two new simple constructive lower bounds for the specific problem at hand (described in Section 3.1). More particularly, we extend two well-known procedures originally proposed for the RCPSP: the Capacity Bound (see, for instance Klein, 2000) and the Critical Sequence Lower Bound (cf. Stinson et al., 1978), to cope with the existence of skills in this problem setting.

The motivation to adapt such methods to the MSRCPSP is threefold: (i) these are classical lower bound methods for the RCPSP whose extension to the MSRCPSP has not been attempted before, to the best of the author's knowledge; (ii) they are very efficient regarding the computational time necessary for their computation; (iii) their extension to the MSRCPSP may open directions for the development of other constructive lower bounds that only make sense when multi-skill resources are involved in the project.

We present below the extensions of those two mechanisms to our problem setting and describe in detail the assumptions made for their generalization.

Skill-based Capacity Bound — *SbCB*

The Skill-based Capacity Bound (*SbCB*) consists of an adaptation of the capacity bound proposed for the RCPSP (see, for instance Klein, 2000). Conversely to the lower bound provided by the length of the critical path, the *SbCB* discards precedence relations between pairs of activities. The *SbCB* determines a lower bound on the objective value of the problem by computing, for each skill, the amount of time (rounded up to the nearest integer whenever its fractional part is greater than 0[1]) that would be required by the resources mastering that skill to process all the requirements of such skill over the whole processing times of all the activities that require it. After performing these operations, we will have L numbers, one for each skill. Since a project is only concluded after the requirements of all skills have been fulfilled, the *SbCB* retrieves the maximum among these values.

We compute this bound as follows:

$$SbCB = \max \left\{ \left\lceil \frac{\sum_{j \in V^l} (p_j \times r_{jl})}{\left|\mathcal{R}^l\right|} \right\rceil \;\middle|\; l = 1, \ldots, L \right\},$$

where $\mathcal{R}^l$ is the set of resources mastering skill $l \in \mathcal{L}$ and set V^l contains all activities that require skill $l \in \mathcal{L}$ for their execution.

Besides discarding precedence relations between pairs of activities, this bound also allows a continuous resource allocation, i.e., it allows resources to be assigned fractionally to the requirements of the activities. In fact, it computes the quotient of the sum of all the requirements for a given skill across all the activities that needed it and the number of resources that master that skill. Moreover, the *SbCB* disregards that an activity has to be in progress for its whole processing time. By computing the product of the processing time of an activity and its requirements for a given skill, the *SbCB* assumes that those requirements could all be met at the expense of, at the limit, 1 time unit, when the number of resources is sufficiently large. To better illustrate this fact, we present an example below.

[1] we recall that in our problem the processing times of the activities and consequently the makespan are positive integers.

Example 4.1 *Consider one activity with processing time of 2 time units and a requirement of 3 resource units for a given skill. If we assume that there are 6 resources mastering that skill, the SbCB would output the value of 1 time unit, hence violating the condition that imposes that every activity has to be in progress for its whole processing time (which is 2 time units, in this case). Furthermore, the SbCB does not verify whether there are enough resources to process an activity. In order to illustrate that statement, let us now consider that there are only 2 resources (instead of 6) mastering the skill needed by such activity. In this situation, the SbCB would retrieve the value of 3 time units. Note that, in this case, the instance would be infeasible, since activity 2 still requires 3 resources to meet its demand for the only skill it needs for being executed.*

Using the data from the example of the problem (Example 3.1) presented in Section 3.1, we compute this bound (using information from Tables 3.1 and 3.2) as follows.

$$SbCB_{l=1} = \left\lceil \frac{3\times2+1\times2+1\times1+2\times1}{3} \right\rceil = \lceil 3,667 \rceil = 4$$

$$SbCB_{l=2} = \left\lceil \frac{3\times3+2\times1+1\times1+1\times2}{4} \right\rceil = \lceil 3,5 \rceil = 4$$

$$SbCB = \max\{SbCB_{l=1}, SbCB_{l=2}\} = 4$$

The value provided by SbCB assures that the project associated with Example 3.1 lasts 4 or more time units.

Skill-based Critical Sequence Lower Bound — *SbCSLB*

By combining both the notion of critical path with the limited availability of the resources, Stinson et al. (1978) developed the so-called Critical Sequence Lower Bound (*CSLB*) for the RCPSP.

We denote the extension of this procedure to MSRCPSP as Skill-based Critical Sequence Lower Bound, abbreviated as *SbCSLB*. The main difference between the *SbCSLB* and the original method relies on the procedure for checking the compatibility of a pair of activities.

The *SbCSLB* begins by scheduling only the activities in the critical path, sequentially, without violating precedence constraints. At this point, we do not assign any resources to meet the skill demands of these activities. Nonetheless, we know that a feasible resource assignment to that schedule exists since, as discussed in Section 3.4.3, a fundamental condition for an instance of the MSRCPSP (and also to the RCPSP) to be feasible is the existence of resources to process each activity individually.

Then, we iteratively select one activity j not belonging to the critical path and insert it into the schedule consisting uniquely of the activities in the critical path. More specifically, for each time $t \in \{ES_j, \ldots, LF_j\}$, an auxiliary graph analogous to the one presented in Figure 3.4 is built. In such graph, the set of nodes $\mathcal{Z}_{W_t}$ contains all the existing resources that may process at least one skill required by either j or the activity in the critical path in progress at that time; the set of nodes $\mathcal{L}_{W_t}$ contains the unique skills required by those activities and the arcs (l, v_s), $l \in \mathcal{L}_{W_t}$ have minimum throughput and capacity equal to the number of resources necessary to execute those two activities simultaneously. If a solution to that problem does not exist for at least p_j consecutive time units (we recall that activity j cannot be preempted) within the processing time-window of activity j (i.e., $t \in \{ES_j, \ldots, LF_j\}$), the difference between the processing time of j and the maximum number of consecutive periods where it can be in progress along with the activities in the critical path is stored in $delta_j$. After performing these operations to all activities not belonging to the critical path, one at a time, the lower bound provided *SbCSLB* is given by the critical path length added to the maximum value of $delta_j$, $j \in V \setminus \{0, n+1\}$.

4.4 Conclusions

In this chapter, we presented several mathematical formulations for the problem being investigated. More particularly, we revisited two optimization models from the literature and proposed two new discrete-time formulations. The main motivation for developing such mathematical models emerged essentially from the need of improving the lower bounds provided by the critical path length, which are usually poor. In fact, from the research concerning the RCPSP, we observed that the discrete-time formulations dominate continuous-time formulations in terms of the optimal value of their corresponding linear programming relaxations. We expect a similar behavior for the MSRCPSP. We revisited several sets of valid inequalities and a reduction test already proposed in the literature for a continuous-time formulation. By substituting the standard precedence constraints (in the discrete-time formulations) by the well-known disaggregated discrete-time precedence constraints, three new discrete-time mathematical models were obtained. One of the aforementioned valid inequalities was discretized in order to be incorporated into the presented discrete-time formulations. Still associated with the need of developing alternative mechanisms to compute lower bounds for the problem being studied, two simple lower bound methods initially proposed for the RCPSP were extended to our problem setting.

CHAPTER 5

Heuristic Algorithms

The multi-skill resource-constrained project scheduling problems can, in theory, be solved by exhaustive search. Nonetheless, such solution technique becomes impracticable due to the enormous computational time required even for small- and medium-sized instances. Despite algorithms performing faster than exhaustive search been already designed for solving the RCPSP, such as the specific branch-and-bound method proposed by Demeulemeester and Herroelen (1997), some reasonably sized instances of that problem remain unsolved. Hence, there is an increased motivation for developing efficient approximate methods for the MSRCPSP being studied, which is a generalized version of the unit-capacity RCPSP.

The most obvious upper bound for any project scheduling problem is $\sum_{j \in V} p_j$. This is the time necessary to execute all the activities in the most adverse case, which corresponds to the full sequential processing of all activities—no activity overlapping exists. Note that this bound is in general also very weak unless, a lower level of parallelization exists in the optimal solution, for instance, due to a high degree of resource scarceness. Correia et al. (2012) propose a simple procedure for obtaining upper bounds on the optimal value of the problem. Although such procedure represents a major improvement over the trivial upper bound mentioned above, the results portrayed by Correia et al. (2012) show that there is much room for improvement in terms of developing efficient procedures for computing tighter upper bounds.

This chapter is devoted to the development of new and more promising approaches to compute feasible solutions to the problem being analyzed. More particularly, two scheduling generation schemes, a multi-pass heuristic and one metaheuristic are proposed.

This chapter is organized as follows. In Section 5.1, the well-known Parallel Scheduling Scheme is extended to our problem setting and a multi-pass constructive heuristic built upon the referred procedure is proposed. In Section 5.2, we extend the well-known Serial Scheduling Scheme to our problem setting and develop a biased random-key genetic algorithm (BRKGA) for the MSRCPSP. The computational tests performed to evaluate the methodological developments proposed in the previous sections along with the mathematical formulations and lower bounds presented in Chapter 4 are reported in Section 5.3.2. This chapter ends with Section 5.4, where an overview of the work done is presented.

5.1 A multi-pass heuristic

One of the most popular constructive heuristics for the RCPSP is the Parallel Scheduling Scheme (PSS) (Brooks and White 1965). This is an iterative algorithm whose maximum number of iterations is equal to the number of activities in the project. In each iteration, and for an appropriate moment in time, a decision set is considered. This set contains all the activities that are eligible to be scheduled, i.e., the activities whose predecessors have had their execution finished, and for which there are available resources to process them. The inclusion of activities in the decision set is performed using the so-called activity *priority values*, such concept was already introduced in Section 3.3.3. These values are assigned to the activities by means of an activity priority rule that, in turn, is a criterion defined for ranking the activities.

The PSS starts by setting a time counter t to zero. Throughout the execution of the algorithm, t will move forward by taking values that depend on the completion times of already scheduled activities. For each value of this counter, a decision set containing all activities that are eligible to be scheduled must be built. The overall procedure can be briefly summarized as follows.

While there are activities in the current decision set remaining to be scheduled, the best ranked activity is selected. That activity is then scheduled to start at time t and the necessary resources are allocated to it. When the decision set at time t is empty or although not empty it is not possible to schedule more activities belonging to this set due to resource constraints, the time counter t is incremented to the minimum completion time of all activities already scheduled; the decision set is then updated, and a new iteration starts. This process is repeated until all the activities in the project have been scheduled. The makespan is computed as the maximum completion time among all activities that do not precede any other.

In this work, we propose a PSS for the MSRCPSP. Since the MSRCPSP is much more complex than the RCPSP due to the multi-skill nature of the resources, a PSS heuristic for the former is more evolved and complex than the one summarized above. In order to cope with that increased complexity, we develop two new features:

i The first feature has the purpose of avoiding iterative activity scheduling and resource assignment. It consists in grouping all the activities that can be scheduled at a specific time t in a set W_t and checking if their skill requirements can be met at once by the available resources.

ii The second feature aims at avoiding random resource selection and allocation. At a specific time t, and given a subset of activities W_t such that there are enough resources to process them simultaneously, there may be more than one subset of resources that meets the requirements of those activities. Hence, it makes sense to determine the best subset of resources with regard to some predefined measure and to efficiently assign these resources to the activities in W_t. With this purpose, we introduce a weight w_k for each resource $k \in \mathcal{R}$, as discussed in Section 3.3.2. We recall that a larger weight indicates a more valuable resource (e.g., masters more skills, masters skills scarcely mastered by the other resources).

In the remainder of this section, we present and discuss in detail the several components of our heuristic. We begin by revisiting the concept of activity priority rules in Section 5.1.1, along with those included in the developed procedure. We proceed analogously for the weights of the resources, whose weight functions considered are presented in Section 5.1.2. Then, in Section 5.1.3, a parallel scheduling scheme heuristic for the MSRCPSP at hand is formally presented. In Section 5.1.4, we discuss the use of the two precedence networks schemes. Finally, in Section 5.1.5, we present the multi-pass version of the referred heuristic.

5.1.1 Priority rules

The use of priority rules is far from new in the area of project scheduling. Furthermore, many rules can be devised. In this work, we considered several well-known rules (Kolisch 1996a, and Demeulemeester and Herroelen 2002) as a starting point for our heuristic.

Table 5.1 summarizes the rules used.

Table 5.1: Activity priority rules.

Abbv.	Rule	Activities sorted by:
LFT	Latest Finish Time	non-decreasing order of their latest finish times
LPT	Longest Processing Time	non-increasing order of their processing times
LST	Latest Start Time	non-decreasing order of their latest start times
GRPW	Greatest Rank Positional Weight	non-increasing order of the sum of their processing times with the processing times of their immediate successors
GRPW*	Greatest Rank Positional Weight*	non-increasing order of the sum of their processing times with the processing times of all their successors
MIS	Most Immediate Successors	non-increasing order of their number of immediate successors
MTS	Most Total Successors	non-increasing order of their total number of successors (immediate and transitive)
SPT	Shortest Processing Time	non-decreasing order of their processing times

Rules *LST* and *LFT* require, respectively, the computation of LS_j and of LF_j of each activity $j \in V$ (cf. Section 3.2 for details regarding their computation). The upper bound used when working with these rules is simply the sum of the processing times of all activities. Regarding for instance, the *LST* rule, the activity with the smallest latest start time receives the best priority value while the activity with the largest latest start time is associated with the worst priority value. In an event of a tie, the activity with the smallest label is selected first. In addition to the "single" activity priority rules just presented, we also consider multi-priority rules. In particular, we consider sequences of two activity priority rules such that the second rule is applied for breaking ties (when they occur). Since we have eight single-priority rules, many multi-priority rules can be defined. In order to keep our analysis more focused, we decided to restrict the number of multi-priority rules considered to *LST* + *GRPW**, *LST* + *LFT*, and *LST* + *MTS*. The reason for this choice will be totally clear in the computational results section and has to do with the fact that these rules yield smaller makespan values.

5.1.2 Resource weights

We introduced the concept of resource weight in Section 3.3. As mentioned in such section, we associate weights to the resources in an attempt to avoid random resource selection and allocation.

We developed three different rules for defining the weight of a resource. The first one, denominated by Resource weight rule 0, is a trivial one that consists in setting the weight of each resource to 0 ($w_k = 0$, $k \in \mathcal{R}$).

The second rule, designated by Resource weight rule 1, sets the weight of each resource to the number of skills it masters:

$$w_k = \left|\mathcal{L}^k\right|, \; k \in \mathcal{R}.$$

The third rule, called Resource weight rule 2, computes a weight of a resource according to the number of skills it masters, the scarceness of each of those skills as well as the aggregated "demand" for that skill. The associated weight function is defined as follows:

$$w_k = \left|\mathcal{L}^k\right| \times \max_{l \in \mathcal{L}^k} \left\{ \frac{K}{\left|\mathcal{R}^l\right|} \times \sum_{j \in V^l} (p_j \times r_{jl}) \right\}, \; k \in \mathcal{R},$$

where set V^l contains the activities requiring skill l ($l \in \mathcal{L}$) and set $\mathcal{R}^l$ contains the resources mastering skill l ($l \in \mathcal{L}$).

We note that the proposed alternatives for associating a weight to the resources are "static" in the sense that the weight associated with each resource can be computed beforehand and remains unchanged during the execution of the heuristic.

5.1.3 Parallel Scheduling Scheme — PSS

As discussed previously, the PSS is an iterative method requiring a number of iterations that in the worst case is equal to the number of activities to be executed in the project. Initially, a time counter is set to 0. During the execution of the procedure, the counter will move forward assuming values that result from the completion times of the activities that are already scheduled.

We begin by discussing the two features introduced at the beginning of Section 5.1 and then we present the developed parallel scheduling scheme heuristic.

Meeting the skill requirements of a set of compatible activities at time t
During the process of building a feasible schedule for the MSRCPSP, when we consider some specific moment in time, the activities can be partitioned into three disjoint sets:

(i) activities already executed.

(ii) activities scheduled but not yet completed.

(iii) unscheduled activities.

Accordingly, at some time t, we can include in a set W_t the unscheduled activities whose predecessors have already been terminated along with those (if there are any still unscheduled) that have no predecessors. If the activities included in W_t are compatible, then they can be processed simultaneously. This occurs if (a) there is no precedence relation between any pair of such activities, and (b) the available resources at time t (resources not assigned to activities with their execution in progress) are enough to fulfill the skill requirements of all activities in W_t simultaneously (at time t).

Since the activities in W_t have no unscheduled predecessors, checking whether W_t is a set of compatible activities reduces to validating condition (b). That is something that can be done in polynomial time by finding whether there is a feasible flow in a specific network $G_{W_t} = (V_{W_t}, E_{W_t})$, as discussed in Section 3.3.1.

It is very important to carefully select the resources, by determining the skills that each of which will be performing and the activity to which they will be assigned. In fact, a proper resource allocation tends to maximize the number of activities that can be started (scheduled) at a given time t.

The proposed procedure goes one step further and, besides verifying whether the resources available at that time are enough to meet all the skill requirements of W_t simultaneously, it associates a weight to each of those resources.

More specifically, the resources are assigned to the skills required by W_t by solving a min-cost flow problem in a network $\widetilde{G}_{W_t} = (V_{W_t}, E_{W_t})$ obtained from $G_{W_t} = (V_{W_t}, E_{W_t})$ by replacing the weight of each arc (v_0, k), $k \in \mathcal{Z}_{W_t}$, by the corresponding resource weight, w_k. That problem is denoted by $MCNFP(\widetilde{G}_{W_t})$. We depict the associated graph in Figure 5.1.

Arcs connecting the source node v_0 with each resource node k and the intermediate arcs (k, l) have capacity equal to 1 to ensure that each resource is selected at most once and thus can contribute with at most one skill.

If no feasible flow exists between v_0 and v_s in the above network, then it is necessary to reduce the cardinality of W_t in order to obtain a compatible set of activities. In this case, we propose removing

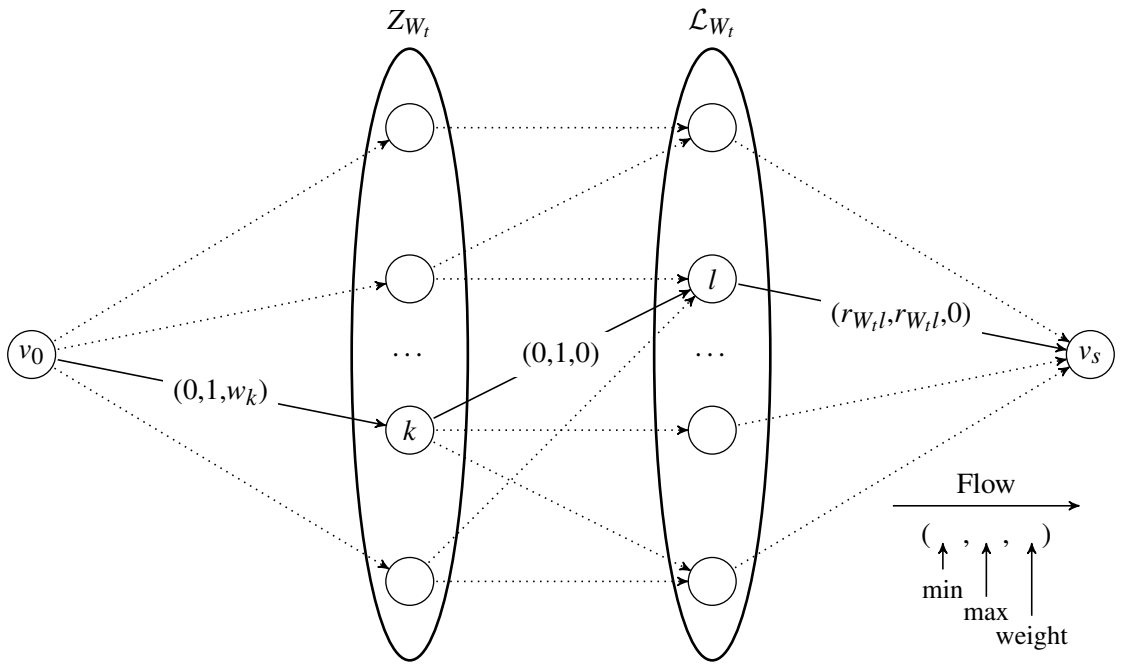

Figure 5.1: A generic $\widetilde{G}_{W_t}$.

from W_t the worst ranked activity, according to the predefined activity priority rule. A decrease in the cardinality of W_t should proceed until the remaining activities in W_t are compatible.

Is is particularly relevant to notice that if the skill requirements of the activities in W_t were to be fulfilled iteratively (i.e., one activity at a time) without accounting for the skill demands of the other activities in W_t, the number of activities scheduled at that time would be potentially smaller and, consequently, the quality of the solution derived could be compromised (as discussed in Example 3.1). By associating weights to the resources (even when they are not considered in the problem data) we are providing some insight regarding their characteristics to the flow problem and hence we are selecting those that, according to our criteria, are the "cheapest" that together fulfill the skill requirements of W_t.

After addressing the problem of determining the resources that fulfill the skill requirements of W_t, we only need to determine the specific activities for which they will be performing the assigned skill. We discuss this topic below.

Resource assignment

Obviously, a feasible flow between v_0 and v_s in $\widetilde{G}_{W_t}$ indicates that W_t is a set of compatible activities.

However, an optimal solution to $MCNFP(\widetilde{G}_{W_t})$ only provides an assignment of the resources $k \in \mathcal{Z}_{W_t}$ to the skills $l \in (\mathcal{L}^k \cap \mathcal{L}_{W_t})$, which is the best one with respect to the resources' weights; it does not indicate the activity each resource is allocated to. In particular, for every $l \in \mathcal{L}_{W_t}$ we obtain a set $\mathcal{X}_{W_tl} \subseteq \mathcal{Z}_{W_t}$—the set of resources assigned to perform skill $l \in \mathcal{L}_{W_t}$ in the solution to the $MCNFP(\widetilde{G}_{W_t})$.

We propose a procedure that heuristically assigns the resources with larger weights to the activities with smaller processing times (as much as possible) in an attempt to make those resources (looked as valuable) free as soon as possible and thus available to be assigned to other activities that may need them. Algorithm 5.1 formalizes the allocation of resources to the (compatible) activities in W_t.

The algorithm determines the assignment of resources $k \in \mathcal{X}_{W_tl}$ to the activities in W_t according to the skills they were assigned to perform in the solution of the corresponding problem $MCNFP(\widetilde{G}_{W_t})$. In each step of this algorithm, the activity j^* in W_t associated with the smallest processing time p_{j^*}, is selected and its corresponding start time, S_{j^*} is set to t (lines 2–4). Afterwards, the requirements of each skill $l \in \mathcal{L}_{j^*}$ are fulfilled through the assignment of the r_{j^*l} resources with the largest weight val-

ues among the ones that belong to $\mathcal{X}_{W_t l}$—the set of resources assigned to l in the optimal solution of $MCNFP(\widetilde{G}_{W_t})$, and which have not been allocated yet (lines 5–13). After meeting all the skill requirements of j^*, this activity is removed from W_t and UV (lines 14–15). The algorithm proceeds until W_t becomes empty.

Algorithm 5.1: Resource assignment

Data: $UV, W_t, \mathcal{X}_{W_t l}, \mathcal{L}_j, p_j, r_{jl}, w_k \quad j \in W_t, l \in \mathcal{L}_j, k \in \mathcal{X}_{W_t l}$
Result: Resources are assigned to the compatible activities in W_t

```
1  begin
2      while W_t ≠ ∅ do
3          Find j* : p_j* = min {p_j : j ∈ W_t};
4          S_j* ⟵ t;
5          for l ∈ L_j* do
6              nra ⟵ 0        // counter for the number of resources already assigned to perform skill l for activity j*;
7              while nra < r_j*l do
8                  Find k* : w_k* = max {w_k : k ∈ X_W_t l};
9                  Assign resource k* to perform skill l for activity j* and set it busy within
                     {S_j*, ..., S_j* + p_j* − 1};
10                 nra = nra + 1;
11                 X_W_t l ⟵ X_W_t l \ {k*};
12             end
13         end
14         W_t ⟵ W_t \ {j*};
15         UV ⟵ UV \ {j*};
16     end
17 end
```

A PSS for the MSRCPSP

After having introduced the elements of the new heuristic approach that we propose for the MSRCPSP, we can now formalize it.

In Algorithm 5.2 we present the pseudo-code of the proposed parallel scheduling scheme. The weight of each resource should be computed according to the chosen resource weight rule and the priority value of each activity should be computed according to the selected activity priority rule. We note that such calculations are performed beforehand and consist of some of the input data required by the developed PSS heuristic.

The proposed PSS starts with the initialization of the relevant parameters (lines 2–5). The set of unscheduled activities, UV, is initialized with all activities (apart from the dummies). The main step of the PSS (lines 6–27) is executed while there are unscheduled activities.

In each iteration, which is associated with a specific moment in time, t, the set W_t is loaded with all the activities that either have no predecessors or whose all predecessors have already finished at that time. If W_t is empty, the value of t is incremented to the next moment in time in which some activity becomes available for execution due to the completion of all its predecessors. Otherwise, either there are enough resources among those available at time t to meet all the skill requirements of the activities in W_t, and hence, all these activities start being processed at time t, or this is not the case and thus the activity with the worst priority value is successively removed from W_t until the resulting set is either empty or all the skill requirements of its activities can be met. When W_t is a set of compatible activities, the former case, Algorithm 5.1 is executed in order to assign the resources in the optimal solution of $MCNFP(\widetilde{G}_{W_t})$ to these activities. A value for the makespan is retrieved after all activities have been scheduled.

Algorithm 5.2: A Parallel Scheduling Scheme (PSS) for the MSRCPSP

Data: $V, E, Pred(j), Succ(j), \mathcal{R}, \mathcal{L}, \mathcal{L}_j, \mathcal{L}^k, p_j, r_{jl}, pv_j, w_k : \quad j \in V, k \in \mathcal{R}, l \in \mathcal{L}_j$
Result: *makespan*

begin
  $UV \longleftarrow V \setminus \{0, n+1\}$;
  $t \longleftarrow 0$;
  $S_0 \longleftarrow 0$;
  $S_j \longleftarrow \infty, j \in UV$;
  while $UV \neq \emptyset$ **do**
    $W_t \longleftarrow \emptyset$;
    for $j \in UV$ **do**
      if $t \geq \max\{S_i + p_i : i \in Pred(j)\}$ **then**
        $W_t \longleftarrow W_t \cup \{j\}$
      end
    end
    while $W_t \neq \emptyset$ **do**
      Build sets Z_{W_t} and $\mathcal{L}_{W_t}$ and solve the $MCNFP(\widetilde{G}_{W_t})$;
      if *a feasible solution to the* $MCNFP(\widetilde{G}_{W_t})$ *exists* (W_t *is a set of compatible activities*) **then**
        for $l \in \mathcal{L}_{W_t}$ **do**
          get $\mathcal{X}_{W_t l}$;
        end
        Execute Algorithm 5.1 to meet the skill requirements of activities $j \in W_t$
      else
        Find $j' \in W_t : pv_{j'}$ is the worst activity priority value;
        $W_t \longleftarrow W_t \setminus \{j'\}$
      end
    end
    $t \longleftarrow \min\{S_j + p_j : j \in V \wedge j \notin UV\}$;
  end
  $makespan \longleftarrow \max\{S_j + p_j : j \in V\}$
end

5.1.4 Precedence network schemes

The PSS just presented can also be applied to the problem obtained by reversing all the arcs in the original (AON) precedence network, hereafter denoted by reversed precedence network. This problem is equivalent to planning the project backwards starting at the end and moving regressively to the beginning—*backward planning*. This is accomplished by considering the last activities as the first ones to be scheduled. Naturally, the problem to be solved is the same but the order by which each activity is scheduled may be different, and hence a different resource selection and assignment may occur, which results in a schedule with, possibly, a different makespan. This concept of backward planning has already been applied to the RCPSP by Li and Willis (1992), Özdamar (1999), Klein (2000), and Alcaraz and Maroto (2001), to mention a few.

5.1.5 The heuristic

The heuristic formalized above is a *single-pass heuristic* since it is only executed once. An execution of the referred single-pass heuristic is associated with one activity priority rule, one resource weight rule and one precedence network scheme. Nonetheless, we can execute the proposed procedure multiple times. By combining each one of the 11 activity priority rules presented in Section 5.1.1 with each one of the 3 developed resource weight rules introduced in Section 5.1.2 and with each one of the 2 precedence

network schemes described in the previous section, a total of 66 schedules can be computed. Among the solutions obtained, the one with minimum makespan can be chosen. This process originates a so-called *multi-pass heuristic*. We formalize such procedure in Algorithm 5.3. The associated computational results are reported in Section 5.3.1.

Algorithm 5.3: A multi-pass PSS heuristic

Data: $V, E, Pred(j), Succ(j), \mathcal{R}, \mathcal{L}, \mathcal{L}_j, \mathcal{L}^k, p_j, r_{jl} \quad j \in V, l \in \mathcal{L}_j, k \in \mathcal{R}$
Result: $makespan^*$
$makespan^* \longleftarrow \infty$;
for *each resource weight rule* **do**
  for $k \in \mathcal{R}$ **do**
    Compute w_k according to the resource weight rule adopted;
  end
  for *each activity priority rule* **do**
    for $j \in V$ **do**
      Compute pv_j according to the predefined activity priority rule;
    end
    for *each precedence network scheme* **do**
      if *reversed precedence scheme* **then**
        Reverse arcs in the precedence network;
      end
      Execute Algorithm 5.2 (PSS);
      if $makespan < makespan^*$ **then**
        $makespan^* \longleftarrow makespan$;
      end
    end
  end
end

This multi-pass heuristic was published in Almeida et al. (2016b).

5.2 A Biased Random-key Genetic Algorithm — BRKGA

Despite the significant advances represented by the work presented in the previous section, much work still remains to be done namely in terms of improving the quality of the feasible solutions obtained. The work presented in the current section emerges in this context. In particular, the new contributions are twofold: (i) we propose a constructive heuristic for the MSRCPSP based on the well-known serial scheduling scheme; (ii) we develop a biased random-key genetic algorithm (BRKGA) to coordinate the approximate method referred in (i) and the parallel scheduling scheme already proposed in Section 5.1.3.

A BRKGA is a population based metaheuristic where each individual is represented by an array of real numbers (the *keys*) in the interval [0, 1] (Gonçalves and Resende, 2011). This method has many similarities with classical genetic algorithms but it uses some different strategies which aim at overcoming a major drawback found in those algorithms. In particular, depending on the way a solution is coded as well as on the crossover operator considered, a genetic algorithm does not necessarily guarantee the feasibility of an offspring when crossing two feasible solutions. In the context of BRKGA, a "child" solution is feasible if the "parents" are feasible. To the best of the author's knowledge, a BRKGA has never been proposed for a multi-skill resource-constrained project scheduling problem although it has been applied successfully to resource-constrained project scheduling problems (see Gonçalves et al., 2008, Mendes et al., 2009, and Gonçalves et al., 2011). BRKGA algorithms have also been successfully

applied to other optimization problems such as packing (Gonçalves and Resende, 2013), facility layout (Gonçalves and Resende, 2015), capacitated minimum spanning trees (Ruiz et al., 2015), among others.

The remainder of this section is organized as follows. In Section 5.2.1, we introduce a generic BRKGA framework and its proprieties. Then, in Section 5.2.2, the problem-dependent components of the BRKGA are presented and discussed in detail. These include the definition of the chromosome structure and the decoding mechanisms.

5.2.1 Preliminaries

A Genetic Algorithm (GA) (Holland, 1975, and Goldberg, 1989) is a metaheuristic that uses concepts of evolution and hereditary to look for good feasible solutions to optimization problems. In a GA, a population of individuals evolves over a number of iterations, called generations, until the defined stopping criteria are met. Each individual is represented by a so-called chromosome and encodes a solution to the problem. A chromosome is represented by an m-dimensional vector of genes (m being a problem-specific number) whose values are referred to as alleles. By using a deterministic problem-dependent algorithm called decoder each chromosome is converted into a solution to the optimization problem and its fitness value can be computed. The quality of the solution a chromosome encodes is given by that value. In each iteration, or generation, the chromosomes that represent the current population produce offspring by means of so-called crossover and mutation operators.

In spite of the large success of GAs, finding the best (or a good) encoding for a particular problem can be a very difficult task. Moreover, crossing two chromosomes (inducing feasible solutions) does not lead necessarily to a feasible offspring. That calls for (possibly much) additional computational effort in order to recover feasibility. With the purpose of overcoming these drawbacks, Bean (1994) introduced the random-key genetic algorithms (RKGA) in the context of sequencing problems. In a RKGA, independently from the optimization problem to be solved, a chromosome is represented by an m-dimensional vector of real numbers in the interval $[0, 1]$—the random keys. In other words, in a RKGA, each allele becomes a random number in interval $[0, 1]$.

Apart from the differences just presented, the RKGA differs from the classical GA in the way the population evolves, namely: (i) it follows an elitist strategy where the chromosomes associated with the best fitness values in one iteration (elite population) are copied unchanged to the next iteration; (ii) it introduces the concept of immigration, associated with the inclusion, in each iteration, of a percentage of new randomly generated chromosomes, called mutants, instead of applying the classical mutation operator; (iii) in order to generate a new chromosome, two parents are randomly selected from the whole population, and are then combined using parameterized uniform crossover (see Spears and Jong, 1991 for further details).

A Biased Random-Key Genetic Algorithm (BRKGA) (Gonçalves and Resende, 2011) differs from a RKGA in the way parents are selected for generating new individuals. In a BRKGA, one parent is selected from the set of elite solutions while the other is selected from the set of non-elite solutions. The process can be described as follows. Let $\rho_e > 0.5$ denote the probability of a descendant inhering an allele from its elite parent. Let c_1, c_2, c_3 be m-dimensional vectors representing, an elite parent, a non-elite parent, and an offspring, respectively. After randomly generating m numbers u_i ($i = 1, \ldots, m$) according to a uniform distribution in the $[0, 1]$, each allele $c_3[i]$ takes the value $c_1[i]$ if $u_i < \rho_e$ and takes the value $c_2[i]$ otherwise.

Algorithm 5.4 depicts a generic BRKGA. In this algorithm, p denotes the number of chromosomes in the population; m is the number of genes in each chromosome; p_e is the percentage of elite chromosomes in the population; p_m denotes the percentage of mutants introduced in each generation; ρ_e represents the probability of a descendant inheriting an allele from its elite parent; $U[0, 1]$ represents a number randomly generated according to a continuous uniform distribution in the interval $[0, 1]$; g is a generation

counter; finally c^* and f^* denote, respectively, the best chromosome and its corresponding fitness value. Some examples of stopping criteria often considered are: a predefined number of generations, a predefined time limit, a predetermined number of generations after the generation where the current best fitness value was found, etc.

Algorithm 5.4: Biased Random-Key Generic Algorithm (BRKGA)

Data: p, p_e, p_m, m, ρ_e
Result: c^*, f^*
begin
  Generate initial population P_1 with p chromosomes where each allele is $U[0, 1]$;
  Compute the fitness of the p chromosomes using the decoder;
  Initialize f^* and c^*;
  $g \longleftarrow 1$;
  while *the stoping criteria are not satisfied* **do**
    Save in the set P_g^e the $\lceil p_e \times p \rceil$ most fit chromosomes of P_g ;
    Copy P_g^e into P_{g+1};
    Generate $\lceil p_m \times p \rceil$ mutants, compute their fitness using the decoder, and copy them to P_{g+1};
    for $j = 1$ *to* $(p - \lceil p_e \times p \rceil - \lceil p_m \times p \rceil)$ **do**
      Randomly select parent c_1 from P_g^e;
      Randomly select parent c_2 from $P_g \setminus P_g^e$;
      for $i = 1$ *to* m **do**
        $u = U[0, 1]$;
        if $u < \rho_e$ **then**
          $c_3[i] \longleftarrow c_1[i]$;
        else
          $c_3[i] \longleftarrow c_2[i]$;
        end
      end
      Compute the fitness of c_3, using the decoder, and copy c_3 to P_{g+1};
    end
    $g \longleftarrow g + 1$;
    if *a better chromosome was found* **then**
      Update f^* and c^*;
    end
  end
end

The size of elite and mutant populations in Algorithm 5.4 are $\lceil p_e \times p \rceil$ and $\lceil p_m \times p \rceil$, respectively. The values taken by p_e and p_m together with all the values taken by the other considered parameters need to be fine-tuned for the particular problem at hand. We note that, despite the size of the elite and mutant populations being rounded up, the summation of the values considered for both of them in the computational experiments will always be smaller than the size of the whole population.

5.2.2 Extension to the MSRCPSP

In order to apply Algorithm 5.4 to the MSRCPSP we only need to define the components of the algorithm that are specific to our problem namely, the structure of the chromosomes and the decoders. Since the chromosomes also contain decoder related information, we begin by introducing the decoder mechanisms to be used and then we concentrate on describing the structure of the chromosomes.

Decoders (constructive heuristics)
Being the MSRCPSP an extension of the RCPSP, two natural constructive heuristics for the former

emerge from extending the well-known parallel and serial scheduling schemes designated by PSS and SSS, respectively (cf. Kolisch, 1996b). Although the PSS and the SSS can be looked at as simple heuristic strategies for the RCPSP, their extension to our problem entails an increased complexity that is mainly related to the selection and assignment of the multi-skill resources to the activities comprising the project.

Since we have already proposed an extension of the PSS to the MSRCPSP in Section 5.1.3, we now focus on extending the SSS to our setting. These two schedule generation schemes render important mechanisms for the success of the new heuristic proposed in this section.

Serial Scheduling Scheme — SSS

The SSS that we propose for the MSRCPSP is also an iterative method that starts by setting to 0 the time counter, again denoted by t. In each iteration, the activity j^* with the best priority value and whose predecessors have already been executed is selected to be scheduled. The time counter, t, is then set to the maximum completion time across all the predecessors of j^*. Activity j^* is scheduled to start at time t if the resources available from time t to time $t + p_{j^*} - 1$ are enough to fulfill all its skill requirements; otherwise, t, is moved forward to the next completion time of the activities already scheduled. Again, the availability of resources is checked. This process repeats until eventually activity j^* is scheduled. In the SSS, the set W_t contains only one (unscheduled) activity: the activity selected to be scheduled next.

The scheme we propose is detailed in Algorithm 5.5. After the initialization of the algorithm (lines 2–5), the main loop starts. Next, the activity j^* with the best priority value among the ones that either have no predecessors or have all their predecessors already scheduled is chosen (line 7). The time t is then set to the earliest precedence feasible start time of activity j^* (lines 8–13). Afterwards, several operations are performed to include in $\mathcal{Z}_{W_t}$ all the resources available in every time $\{t, \ldots, t + p_{j*} - 1\}$ that master at least one skill required by j^* (lines 17–22). Only these resources are eligible to perform a given skill during the whole processing time of j^*. After solving the induced problem $MCNFP(\widetilde{G}_{W_t})$, two situations may arise: an optimal solution is found or no feasible flow exists. In the former case, the unique activity j^* in W_t is scheduled to start at time t (line 25) and the set $\mathcal{X}_{W_t l}$, ($l \in \mathcal{L}_{j^*}$) indicates which resources are assigned to skill $l \in \mathcal{L}_{j^*}$ (line 26). In the latter situation, t is incremented to the minimum completion time across all the activities that are already scheduled (line 30).

In contrast to the PSS, in the SSS that we have just proposed, it is possible to have already scheduled activities that start after t. In fact, when an activity, say j^*, is selected to be scheduled, the time t is moved to the maximum completion time of all predecessors of j^*. Therefore, it is possible to have other already scheduled activities which have the necessary resources allocated and their start times higher than t. Hence, for a given time t, deciding whether activity j^* can start being processed at that time requires checking if the resources available in every time slot where j^* will be in progress, i.e., from t to its provisional finish time $t + p_{j^*} - 1$, can fulfill all its skill requirements.

Chromosome

The structure of the chromosomes that we propose is inspired on the chromosomes considered by Gonçalves et al. (2008), Mendes et al. (2009) and Gonçalves et al. (2011) for resource-constrained project scheduling problems, which encode only activity-related information.

We define a chromosome as having $m = n + K + 2$ genes, where n is the number of non-dummy activities in the project and K is the number of resources. In this chromosome, the alleles, u_i, $i = 1, \ldots, m$, are random numbers generated in the interval $[0, 1]$. The last 2 genes are associated with the decoding mechanism to be used for transforming a chromosome into a feasible solution to the problem. In particular, one of these genes indicates the scheduling generation scheme to be employed (parallel or serial) while the other refers to the precedence network scheme to be considered (original or reversed)—already discussed in Section 5.1.4. The chromosome structure we are proposing is illustrated in Figure 5.2.

Algorithm 5.5: A Serial Scheduling Scheme (SSS) for the MSRCPSP

Data: $V, E, Pred(j), Succ(j), \mathcal{R}, \mathcal{L}, \mathcal{L}_j, \mathcal{L}^k, p_j, r_{jl}, pv_j, w_k : j \in V, k \in \mathcal{R}, l \in \mathcal{L}_j$
Result: *makespan*

begin
  $UV \longleftarrow V \setminus \{0, n+1\}$;
  $t \longleftarrow 0$;
  $S_0 \longleftarrow 0$;
  $S_j \longleftarrow \infty, j \in UV$;
  while $UV \neq \emptyset$ **do**
    Find $j^* \in UV : Pred(j^*) \cap UV = \emptyset \wedge pv_{j*}$ is the best activity priority value across all unscheduled activities;
    if $Pred(j^*) = \emptyset$ **then**
      $t \longleftarrow 0$;
    else
      $t \longleftarrow \max\{S_i + p_i : i \in Pred(j^*)\}$;
    end
    Compute $\mathcal{R}_{j^*} = \{k \in \mathcal{R} \ : \ \mathcal{L}^k \cap \mathcal{L}_{j^*} \neq \emptyset\}$ // resources with skills required by activity j^*;
    $W_t \longleftarrow \{j^*\}$;
    while j^* *unscheduled* **do**
      $\mathcal{Z}_{W_t} \longleftarrow \emptyset$;
      for $k \in \mathcal{R}_{j^*}$ **do**
        if k *is available in every time instant* $\{t, ..., t + p_{j^*} - 1\}$ **then**
          $\mathcal{Z}_{W_t} \longleftarrow \mathcal{Z}_{W_t} \cup \{k\}$;
        end
      end
      Solve the $MCNFP(\widetilde{G}_{W_t})$;
      if $MCNFP(\widetilde{G}_{W_t})$ *has an optimal solution* **then**
        $S_{j^*} \longleftarrow t$;
        For each skill $l \in \mathcal{L}_{j^*}$, assign the resources $k \in \mathcal{X}_{W_t l}$ and set them busy within $t \in \{S_{j^*}, \ldots, S_{j^*} + p_{j^*} - 1\}$;
        $UV \longleftarrow UV \setminus \{j^*\}$;
      else
        $t \longleftarrow \min\{S_u + p_u : u \notin UV \wedge S_u + p_u > t\}$ // increment t;
      end
    end
  end
  $makespan \longleftarrow \max\{S_j + p_j : j \in V\}$;
end

It is worth noticing that a chromosome structure with a gene indicating whether a parallel or a serial scheduling scheme is used has already been considered for the RCPSP by Hartmann (2002). Furthermore, the solution encoding proposed by Alcaraz and Maroto (2001) also includes one additional gene for determining whether the original or reversed precedence network is used. To the best of the author's knowledge, a chromosome structure that includes genes associated with the resources along with a gene for decoder selection and a gene for encoding the precedence network scheme has never been attempted before.

We recall that we have already presented a parallel scheduling scheme and a serial scheduling scheme for computing feasible solutions to the MSRCPSP. Similarly to the PSS proposed previously, the SSS can also be applied to either the original or the reversed precedence network (cf. Section 5.1.4). Nevertheless, these heuristics also require the assignment of a priority value, pv_j, for each activity $j \in V \setminus \{0, n+1\}$ as well as a weight value, w_k, to each resource $k \in \mathcal{R}$. In our BRKGA, this information will be embedded in

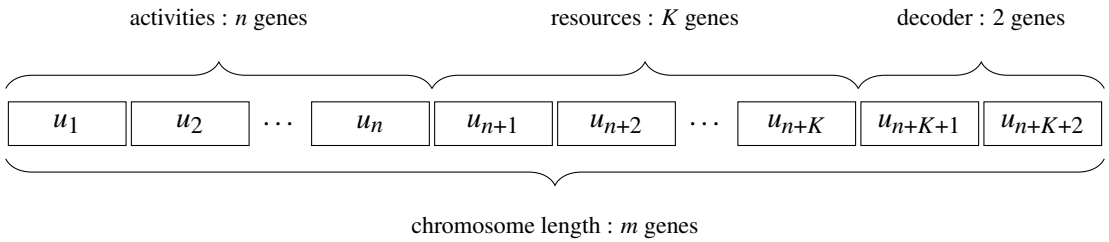

Figure 5.2: A generic chromosome for the BRKGA.

the chromosomes since it seemed more advantageous to rely on the evolutive framework of the algorithm for adapting this data throughout its execution, instead of using static predefined values. In particular, the value of each one of the first n genes will give the priority value of the corresponding activity, that is: $pv_i = u_i$, $i = 1, \ldots, n$. Additionally, the values of the next K following genes will give the weights of the resources, i.e., $w_k = u_{n+k}$, $k = 1, \ldots, K$.

The last 2 genes of a chromosome are associated with the decoder. The information provided by these genes is summarized in Table 5.2.

Table 5.2: Decoder parameters.

gene number	parameter	$0 \leq u_i < 0.5$	$0.5 \leq u_i \leq 1$
$n + K + 1$	scheduling scheme	SSS	PSS
$n + K + 2$	precedence network	original	reversed

Figure 5.3 illustrates a chromosome within our BRKGA. For the first $n+K$ genes of the chromosome, we assume that the larger the values taken by the alleles, the higher the priorities or weights, depending on whether they refer to activities or to resources, respectively. From the first n genes of the chromosome depicted in Figure 5.3, we observe that the priority value of activity 1 is higher than the priority value of activity 2. With regard to the resources, we observe that if resources 1 and 2 are involved in a solution to a min-cost flow problem $MCNFP(\widetilde{G}_{W_t})$ and are both assigned to the same skill $l \in \mathcal{L}_{W_t}$, resource 1 is first attempted to be assigned to the activities $j \in W_t$ with shorter processing times, since it is associated with a larger weight than resource 2. The alleles $n + K + 1$ and $n + K + 2$, indicate, respectively, that the parallel scheduling scheme should be considered and applied to the reversed precedence network.

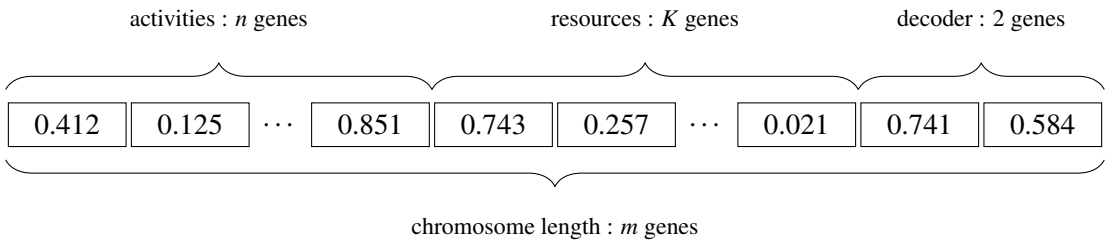

Figure 5.3: Chromosome: example

With this chromosome structure, the same priority values (obtained from the first n genes) and the same weight values (obtained from the following K genes) may originate four different feasible solutions to the MSRCPSP (two possible precedence network schemes—original and reversed, combined with two decoders—PSS and SSS). This kind of structure may help diversifying the exploration of the solution space. Another positive feature of this representation is its versatility. In fact, if the values of the last two genes are fixed for all the chromosomes, this means that all the chromosomes will be decoded with the same algorithm. Accordingly, we may adjust the values of one or both of these genes, if we want to

analyze (or impose) some specific configuration.

The work developed in this section was included in the working paper by Almeida et al. (2016a), which is currently submitted for publication.

5.3 Computational experiments and results

In this section, we report the numerical experiments performed on the developments presented in this thesis. The multi-pass heuristic and the BRKGA were coded in C++ programming language. The mathematical formulations and the min-cost flow problems were solved by integrating IBM ILOG CPLEX 12.6 with C++ through Concert Technology. Regarding the mathematical models, apart from the time limit (that we specify later in this section), all other CPLEX parameters were kept at their default values. All computational experiments were performed on a machine running an Intel Core i7 4770K with 32GB of RAM.

The computational experiments were performed on two different sets of instances from the literature: *Set 1* (see Section 3.4.2) and *Set 2* (see Section 3.4.4).

This section is organized as follows. In Section 5.3.1, we present the computational results of the proposed multi-pass heuristic. In the following section, Section 5.3.2, we discuss the methodology for selecting the values for the parameters of the developed BRKGA, thus fine-tunning the algorithm. Then, we report the numerical results of the BRKGA according to values derived for its parameters. The results provided by the mathematical formulations and lower bound methods (presented in Chapter 4) are reported in Section 5.3.3.

5.3.1 Multi-pass heuristic

A thorough computational study of the multi-pass heuristic presented in Section 5.1 was performed on two sets of instances: *Set 1* and *Set 2*. The corresponding results are presented and discussed in this section. We first concentrate on the small instances and then we move to the larger ones.

Set 1

As we have mentioned before, the instances in *Set 1* were generated and worked out by Correia et al. (2012). In particular, for these instances and due to that work, we know the optimal value for 203 out of the 216 instances and for the other ones we have a lower bound, which is the best one found by Correia et al. (2012). These optimal values and lower bounds can be used to compute the percentage gap (or an upper bound on that gap) of the solution values obtained by the multi-pass heuristic proposed in this thesis. This is done according to

$$gap = \frac{Z^H - Z^{LB}}{Z^{LB}} \times 100\%,$$

where Z^H denotes the upper bound provided by the heuristic and Z^{LB} denotes the optimal value or the best known lower bound obtained by Correia et al. (2012).

We start by analyzing separately the behavior of the different activity priority rules studied. In addition to the single-priority rules summarized in Table 5.1, we also considered three multi-priority rules, namely *LST* + *GRPW**, *LST* + *LFT*, and *LST* + *MTS*. The choice made in terms of multi-priority rules resulted from a few preliminary tests conducted. The preliminary tests made it clear that, on average, the rule *LST* has the best performance, followed by *GRPW**, *LFT*, and *MTS*, which were respectively the second, third and fourth rules that performed best (on average). Accordingly, since it was not reasonable to present results for all possible multi-priority rules, we focused the analysis on combining the one that performed best in the preliminary tests with the second, third and fourth best in terms of average perfor-

mance. Table 5.3 contains the results obtained. In this table, the instances are grouped according to *SF*, *NC*, and *MRS*. Each row in the table refers to average results for 6 instances.

Table 5.3: Multi-pass heuristic — *Set 1*: average gaps for the activity priority rules studied.

SF	*NC*	*MRS*	LFT	LPT	LST	MIS	MTS	SPT	GRPW	GRPW*	LST + GRPW*	LST + LFT	LST+ MTS	Average
1	1.5	0.1250	13.03	16.78	11.82	15.31	11.47	21.39	19.14	10.79	11.82	11.17	11.82	9.47
		0.1563	5.36	6.87	4.77	4.97	5.56	11.48	5.71	4.37	4.77	4.77	4.77	4.37
		0.1875	0.00	1.71	0.46	1.25	0.32	1.85	1.57	0.78	0.46	0.46	0.46	0.00
	1.8	0.1250	3.14	7.17	2.86	5.34	3.21	9.75	4.69	1.57	2.50	2.86	2.50	1.57
		0.1563	2.68	6.39	3.08	3.45	3.11	11.67	6.44	3.41	2.70	3.08	2.70	1.92
		0.1875	0.46	0.93	0.93	0.93	0.93	4.23	1.69	0.93	0.93	0.46	0.46	0.46
	2.1	0.1250	2.87	4.84	1.07	3.03	1.51	7.94	3.83	1.07	1.07	1.07	1.07	1.07
		0.1563	0.62	0.00	0.00	1.21	0.00	2.17	0.96	0.29	0.00	0.00	0.00	0.00
		0.1875	0.00	0.00	0.00	0.00	0.00	0.81	0.00	0.00	0.00	0.00	0.00	0.00
0.75	1.5	0.1250	8.38	13.15	7.25	11.49	10.40	11.48	10.17	7.48	7.25	8.11	7.48	3.98
		0.1667	3.15	9.42	3.04	4.68	5.26	13.53	4.57	2.16	3.04	2.70	3.04	0.90
		0.2083	4.33	10.37	4.31	5.77	6.26	4.11	5.80	4.33	4.31	4.31	4.31	3.84
	1.8	0.1250	9.52	14.27	8.02	11.93	10.94	16.68	12.71	9.11	9.21	7.70	8.92	6.47
		0.1667	3.50	7.76	2.93	3.86	3.80	9.03	3.50	3.84	2.93	2.93	2.93	2.93
		0.2083	0.00	0.32	0.00	0.00	0.00	0.94	0.32	0.00	0.00	0.00	0.00	0.00
	2.1	0.1250	5.62	9.79	5.61	10.04	5.74	12.24	9.71	5.03	5.25	4.64	5.25	3.99
		0.1667	1.20	3.34	0.00	2.19	0.33	7.26	3.03	1.96	0.73	0.33	0.73	0.00
		0.2083	0.79	2.73	0.79	2.35	1.59	1.19	2.35	0.79	0.79	0.79	0.79	0.79
0.5		0.1250	10.58	21.48	12.06	13.40	12.70	17.04	15.62	13.35	12.50	13.02	12.50	6.17
	1.5	0.1625	5.13	13.63	5.29	10.97	6.10	13.44	16.97	5.29	5.29	4.95	5.29	4.27
		0.1875	3.03	5.43	0.88	5.45	4.80	7.47	2.53	2.02	1.26	1.77	1.26	0.88
	1.8	0.1250	7.73	18.42	8.78	9.98	9.57	15.84	13.24	9.70	8.78	8.44	8.44	5.19
		0.1625	7.13	13.36	3.66	8.42	5.64	11.46	8.31	2.42	3.66	3.66	3.66	2.42
		0.1875	0.36	4.85	0.71	4.15	2.86	9.97	4.01	1.44	0.71	0.71	0.71	0.36
	2.1	0.1250	7.52	16.02	8.59	7.08	6.93	15.56	13.15	6.80	6.79	8.59	6.79	5.00
		0.1625	1.19	3.45	1.25	3.55	3.67	2.76	2.64	2.33	1.25	1.25	1.25	0.93
		0.1875	0.28	0.98	0.42	2.65	0.39	1.91	0.56	0.00	0.42	0.42	0.42	0.00
var.	1.5	0.1250	11.88	21.35	10.61	18.75	15.89	22.59	19.90	10.95	10.61	10.30	10.61	7.55
		0.1667	9.47	10.37	8.62	8.25	10.58	12.77	10.48	8.62	8.62	8.62	8.62	7.41
		0.2083	1.91	2.49	1.42	6.02	1.96	4.55	3.47	1.42	1.42	1.42	1.42	1.42
	1.8	0.1250	7.62	14.43	6.81	8.77	9.03	17.67	9.75	6.13	6.10	6.10	6.10	5.16
		0.1667	4.85	10.17	4.88	5.84	6.90	12.65	10.75	5.30	5.30	4.02	5.30	4.02
		0.2083	1.01	2.34	0.00	2.94	2.44	4.37	2.38	1.43	0.00	0.00	0.00	0.00
	2.1	0.1250	4.91	6.28	4.99	10.21	6.79	10.43	6.34	4.96	4.71	4.99	4.71	2.46
		0.1667	1.81	7.81	1.29	5.18	2.13	7.05	2.24	1.61	1.29	1.29	1.29	0.33
		0.2083	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Average			4.20	8.02	3.81	6.09	4.97	9.31	6.63	3.94	3.79	3.75	3.77	**2.65**

For each activity priority rule, the gap computed for each instance considered the minimum makespan obtained after running the heuristic with each distinct pair of resource weight rule and precedence network scheme, resulting in a total of 6 passes.

The last column of this table contains the best average gaps obtained after running each instance with all possible combinations of the 11 activity priority rules with the 3 resource weight rules and with 2 precedence schemes: original and reversed networks. Therefore, the gap of each instance was computed considering the best makespan obtained after the 66 passes of the heuristic.

Observing Table 5.3 we conclude that there is no activity priority rule performing always better than any other. However, considering the average gaps displayed in the last row, it is possible to identify the single-priority rule *SPT* as the one performing the worst and the multi-priority rule *LST + LFT* as the best.

We deepen our analysis by focusing on the resource weight rule used and also on the use of a forward (original precedence network) or backward (reversed precedence network) mechanism for scheduling the activities. For comparison purposes, we consider the heuristic for the MSRCPSP developed by Correia et al. (2012), which, to the best of the author's knowledge, is the only heuristic that can be found in the

literature for the problem at hand. It is important to note that Correia et al. (2012) put the emphasis of their work on a new mixed-integer linear programming optimization model. Nevertheless, since the authors needed an upper bound on the optimal value of the problem, in order to load some valid inequalities, a very simple heuristic was introduced for generating such upper bound.

The results are presented in Table 5.4. Similarly to the previous table, the instances are grouped according to *SF*, *NC*, and *MRS* and each row in the table refers to the average results for 6 instances. In this table, we find the results organized according to 5 sets of columns: (i) Columns 4–8 for resource weight rule 0, (ii) Columns 9–13 for resource weight rule 1, (iii) Columns 14–18 for resource weight rule 2, (iv) Columns 19 and 20 reflecting the minimum makespan obtained by the multi-pass heuristic consisting of all the combinations of the 3 developed resource weight rules, the 11 activity priority rules and the 2 precedence network schemes (original and reversed), and (v) Columns 21 and 22 for the results associated with the heuristic proposed by Correia et al. (2012).

The first two columns associated with each resource weight rule refer to the results obtained with the multi-pass version of the heuristic when a single precedence network scheme is considered, i.e., for each instance, and each precedence network scheme, the makespan considered (for calculating the associated gap) is the minimum achieved after running the heuristic with the 11 activity priority rules (8 single-priority and 3 multi-priority). More specifically, in Columns 4, 9 and 14, we observe the results of when a forward scheme (original precedence network) is employed whereas Columns 5, 10, and 15 depict the results of when a backward scheme (reversed precedence network) is used.

The third to fifth columns of each resource weight rule (Columns 6–8 for resource weight rule 0, Columns 11–13 for resource weight rule 1 and Columns 16–18 for resource weight rule 2) consider the 11 activity priority rules together with the 2 schemes for scheduling the activities (forward and backward). The gaps were computed using the minimum makespan among 22 values (some possibly equal) obtained. Regarding the CPU time (CPU (sec)) and the time for solving the minimum cost network flow problems (MCNFP (sec)) we consider the sum of the 22 values corresponding to each pass of the heuristic.

All the activity priority rules and resources weight rules were then applied to the original and to the reversed precedence networks and the lowest makespan value was retrieved. The gaps associated with these results are presented in Column 19 and the total average time spent computing all these values is presented in Column 20. The last two columns refer to the average gaps and to the corresponding CPU time provided by the heuristic proposed by Correia et al. (2012).

Finally, the last row of Table 5.4 reports the average values over all the instances in *Set 1*.

A closer look at this table indicates that, on average, resource weight rule 2 outperforms the other two resource weight rules in terms of gap. On the other hand, resource weight rule 0 is the one producing the worst average gaps. The results further reveal that using the reversed precedence network besides the original one allowed small improvements on the average gaps of the resource weight rules. However, it is worth considering both structures (original and reversed) within a heuristic scheme. In fact, the average results obtained for instances with $SF = 1$, $NC = 1.5$, and $MRS = 0.1875$ are examples of where, for resource weight rules 1 and 2, the use of the two precedence network schemes yielded a 0% gap but neither the original network nor the reversed provided a gap of 0% when used alone.

Regarding the average CPU time spent in solving the minimum cost network flow problems, we can observe that it consumes approximately 36% of the total CPU time of the heuristic.

From columns 19 and 20 of Table 5.4, we can observe that by running the 3 resource weight rules, it was possible to get an average gap of 2.65% in less than one second (CPU time). These results represent a considerable improvement when compared to the results provided by the heuristic proposed by Correia et al. (2012), which are depicted in the last two columns of the table.

In order to analyze the influence of *SF*, *NC* and *MRS* on the performance of the heuristic and, in particular, on the behavior of each activity priority rule and resource weight rule, the results presented in Tables 5.3 and 5.4 were grouped according to the values of these parameters and are displayed in

Tables 5.5 and 5.6, respectively.

In both tables and for each parameter value, the best average gap provided by each activity priority rule (Table 5.5) and by each resource weight rule (Table 5.6) is highlighted by boldface.

The results reported in Table 5.5 show that $GRPW^*$ and $LST + MTS$ were the best activity priority rules for instances with $SF = 1$ and $SF = 0.5$, respectively. For the other values of this parameter $LST + LFT$ produced the best average gaps. For $NC = 1.5, 1.8$ and 2.1 the best priority rules were, respectively, LST, $LST + LFT$, and $LST + GRPW^*$ (or $LST + MTS$). In terms of MRS, the multi-priority rules $LST + MTS$, $LST + LFT$ and $LST + GRPW^*$ attained the best average gaps in 4, 3 and 2 different values of MRS, respectively.

In Table 5.6, we observe that resource weight rule 2 yields the best average gaps for (i) 3 out of 4 skill factor values (0.5, 0.75 and var.); (ii) 2 out of 3 network complexity values (1.5 and 2.1); and (iii) 4 out of 6 modified resource strength values (0.1250, 0.1625, 0.1667 and 0.2083). The best average gap for the instances associated with either $SF = 1$ or $NC = 1.8$ or $MRS = 0.1563$ were obtained by resource weight rule 0. Resource weight rule 1 produced the best average gaps for $MRS = 0.1875$ and $MRS = 0.2083$, the latter tied with resource weight rule 2. We observe that a decrease of either the SF or the NC leads to an increase of the average gaps of the multi-pass heuristic proposed. Finally, the value of MRS associated with the smallest gap was 0.1875 followed by the values 0.2083 and 0.1563.

Table 5.4: Multi-pass heuristic — *Set 1*: average gaps for each resource weight rule.

SF	*NC*	*MRS*	Resource Weight Rule 0 - R0					Resource Weight 1 Rule - R1					Resource Weight Rule 2 - R2					Best min(R0, R1, R2)		Correia et al. (2012)	
			OPN	RPN	min(OPN,RPN)			OPN	RPN	min(OPN,RPN)			OPN	RPN	min(OPN,RPN)						
			Gap (%)	Gap (%)	Gap (%)	CPU (sec)	MCNFP (sec)	Gap (%)	Gap (%)	Gap (%)	CPU (sec)	MCNFP (sec)	Gap (%)	Gap (%)	Gap (%)	CPU (sec)	MCNFP (sec)	Gap (%)	CPU (sec)	Gap (%)	CPU (sec)
1	1.5	0.1250	11.47	13.51	9.47	0.390	0.146	12.11	11.70	9.47	0.354	0.125	12.11	11.70	9.47	0.352	0.120	9.47	1.096	35.25	1.159
		0.1563	4.37	4.37	4.37	0.403	0.159	4.37	4.37	4.37	0.372	0.169	4.37	4.37	4.37	0.370	0.136	4.37	1.145	32.35	0.772
		0.1875	0.83	0.32	0.32	0.466	0.161	0.51	0.32	0.00	0.461	0.164	1.62	0.32	0.00	0.448	0.179	0.00	1.374	21.63	0.326
	1.8	0.1250	1.57	2.70	1.57	0.377	0.125	1.57	2.70	1.57	0.348	0.161	1.57	2.70	1.57	0.326	0.135	1.57	1.051	22.28	0.648
		0.1563	3.35	1.92	1.92	0.370	0.140	3.35	2.30	2.30	0.440	0.169	3.79	2.28	2.28	0.386	0.164	1.92	1.195	33.25	0.462
		0.1875	0.93	0.46	0.46	0.476	0.172	0.93	0.46	0.46	0.448	0.170	0.93	0.46	0.46	0.409	0.165	0.46	1.333	23.05	0.206
	2.1	0.1250	2.39	1.96	1.07	0.385	0.116	1.07	1.07	1.07	0.361	0.132	1.07	1.07	1.07	0.357	0.145	1.07	1.103	18.95	0.365
		0.1563	1.40	0.00	0.00	0.387	0.141	1.40	0.71	0.00	0.352	0.133	1.40	0.00	0.00	0.377	0.158	0.00	1.116	16.15	0.240
		0.1875	0.00	0.00	0.00	0.461	0.146	0.00	0.00	0.00	0.450	0.190	0.00	0.00	0.00	0.394	0.169	0.00	1.304	11.49	0.117
0.75	1.5	0.1250	8.58	7.24	6.94	0.338	0.102	7.13	4.59	3.98	0.336	0.107	8.23	4.85	4.25	0.302	0.101	3.98	0.976	31.16	0.956
		0.1667	1.95	3.09	1.28	0.333	0.107	3.06	2.37	1.27	0.323	0.124	2.68	2.37	1.27	0.344	0.148	0.90	1.000	35.54	0.536
		0.2083	4.61	4.82	3.84	0.390	0.137	4.61	4.33	3.84	0.367	0.133	4.31	4.33	3.84	0.388	0.151	3.84	1.145	32.54	0.307
	1.8	0.1250	8.00	8.89	7.46	0.325	0.130	9.49	10.15	8.45	0.305	0.104	9.03	10.52	8.68	0.286	0.117	6.47	0.916	21.14	0.682
		0.1667	2.93	4.97	2.93	0.320	0.118	3.50	4.40	3.50	0.336	0.116	3.50	3.82	2.93	0.297	0.099	2.93	0.952	23.86	0.471
		0.2083	0.77	0.30	0.00	0.375	0.133	1.22	1.80	0.90	0.370	0.122	0.90	1.80	0.90	0.372	0.112	0.00	1.117	20.10	0.253
	2.1	0.1250	4.29	6.16	3.99	0.289	0.076	4.29	5.08	3.99	0.283	0.109	4.87	5.08	3.99	0.295	0.135	3.99	0.867	16.90	0.365
		0.1667	1.19	1.48	0.40	0.362	0.158	1.71	1.13	0.40	0.336	0.113	0.44	2.45	0.44	0.286	0.135	0.00	0.984	36.51	0.234
		0.2083	1.19	2.73	0.79	0.364	0.145	0.79	2.73	0.79	0.377	0.133	0.79	2.73	0.79	0.396	0.167	0.79	1.137	29.86	0.154
0.5	1.5	0.1250	8.01	11.82	7.40	0.268	0.083	8.58	10.66	7.44	0.239	0.068	8.31	8.30	6.80	0.242	0.086	6.17	0.749	18.79	0.464
		0.1625	6.90	6.94	5.27	0.278	0.088	6.53	5.94	5.27	0.255	0.068	5.53	5.94	4.27	0.263	0.109	4.27	0.796	22.27	0.487
		0.1875	2.90	3.41	2.90	0.310	0.115	1.64	2.53	1.26	0.284	0.078	2.40	2.15	1.64	0.297	0.093	0.88	0.890	32.35	0.352
	1.8	0.1250	7.79	10.58	6.24	0.216	0.083	8.44	9.86	6.40	0.234	0.081	7.76	9.86	6.06	0.208	0.060	5.19	0.658	21.51	0.297
		0.1625	7.72	4.59	3.35	0.304	0.071	5.46	4.69	3.15	0.307	0.094	5.46	2.73	2.42	0.260	0.068	2.42	0.872	22.86	0.339
		0.1875	4.40	2.06	0.75	0.289	0.102	3.20	2.49	0.71	0.276	0.088	3.59	2.49	1.06	0.281	0.096	0.36	0.846	32.59	0.354
	2.1	0.1250	7.86	8.07	6.77	0.232	0.110	7.82	6.24	5.99	0.257	0.094	7.13	5.00	5.00	0.224	0.055	5.00	0.713	13.45	0.204
		0.1625	2.78	1.25	1.25	0.263	0.071	2.47	0.93	0.93	0.255	0.083	1.96	0.93	0.93	0.291	0.086	0.93	0.809	21.11	0.208
		0.1875	2.23	2.20	0.97	0.265	0.083	1.91	1.06	0.28	0.258	0.089	1.10	1.06	0.00	0.260	0.071	0.00	0.783	19.28	0.152
var.	1.5	0.1250	12.69	15.53	10.41	0.255	0.091	11.69	11.65	7.55	0.271	0.122	11.07	10.21	7.86	0.239	0.101	7.55	0.765	28.75	0.834
		0.1667	8.25	10.84	8.25	0.341	0.112	8.20	10.36	7.41	0.304	0.092	8.57	11.21	7.78	0.294	0.089	7.41	0.939	34.13	0.624
		0.2083	3.27	1.42	1.42	0.356	0.120	2.73	1.42	1.42	0.359	0.117	2.73	1.42	1.42	0.333	0.104	1.42	1.049	34.49	0.415
	1.8	0.1250	8.07	5.52	5.52	0.297	0.076	8.51	6.51	6.19	0.289	0.102	8.51	5.48	5.48	0.286	0.088	5.16	0.872	24.47	0.766
		0.1667	5.77	6.25	4.02	0.328	0.109	5.36	6.54	4.30	0.341	0.138	5.36	6.54	4.30	0.334	0.086	4.02	1.002	24.52	0.417
		0.2083	2.90	1.74	1.39	0.361	0.143	1.41	0.00	0.00	0.381	0.169	1.91	0.00	0.00	0.351	0.138	0.00	1.093	32.20	0.304
	2.1	0.1250	4.17	4.90	2.78	0.313	0.104	4.16	5.14	3.68	0.310	0.114	4.63	5.14	3.37	0.294	0.144	2.46	0.916	21.04	0.324
		0.1667	4.17	2.72	2.13	0.317	0.153	3.18	1.50	0.33	0.297	0.099	3.18	1.50	0.33	0.297	0.104	0.33	0.911	26.43	0.393
		0.2083	1.00	0.00	0.00	0.346	0.145	1.00	0.00	0.00	0.362	0.136	1.00	0.00	0.00	0.325	0.114	0.00	1.033	19.26	0.164
Average			4.46	4.58	3.27	0.337	0.119	4.26	4.10	3.02	0.330	0.120	4.22	3.91	**2.92**	**0.318**	0.117	**2.65**	**0.986**	25.32	0.426

OPN (RPN): Original (Reversed) Precedence Network; CPU: Total CPU time; MCNFP: Total CPU time spent in solving minimum cost network flow problems.

Table 5.5: Multi-pass heuristic — *Set 1*: average gaps for each activity priority rule, summary.

		LFT	LPT	LST	MIS	MTS	SPT	GRPW	GRPW*	LST + GRPW*	LST + LFT	LST+ MTS	Average
SF	1	3.13	4.96	2.78	3.94	2.90	7.92	4.89	**2.58**	2.69	2.65	2.64	2.10
	0.75	4.05	7.91	3.55	5.81	4.92	8.50	5.80	3.86	3.72	**3.50**	3.72	2.55
	0.5	4.77	10.85	4.63	7.29	5.85	10.61	8.56	4.82	4.52	4.76	**4.48**	2.80
	var.	4.83	8.36	4.29	7.33	6.19	10.23	7.26	4.49	4.23	**4.08**	4.23	3.15
NC	1.5	6.35	11.09	**5.88**	8.86	7.61	11.81	9.66	5.96	5.95	5.97	5.97	4.19
	1.8	4.00	8.37	3.55	5.47	4.87	10.36	6.48	3.77	3.57	**3.33**	3.48	2.54
	2.1	2.23	4.60	2.00	3.96	2.42	5.78	3.74	2.07	**1.86**	1.95	**1.86**	1.21
MRS	0.1250	7.73	13.66	7.37	10.44	8.68	14.88	11.52	7.25	7.22	7.25	**7.18**	4.84
	0.1563	2.89	4.42	2.62	3.21	2.89	8.44	4.37	2.69	**2.49**	2.62	**2.49**	2.10
	0.1625	4.49	10.14	3.40	7.65	5.14	9.22	9.31	3.35	3.40	**3.29**	3.40	2.54
	0.1667	4.00	8.15	3.46	5.00	4.83	10.38	5.76	3.91	3.65	**3.32**	3.65	2.60
	0.1875	0.69	2.32	0.57	2.40	1.55	4.37	1.73	0.86	0.63	0.64	**0.55**	0.28
	0.2083	1.34	3.04	**1.09**	2.85	2.04	2.53	2.39	1.33	**1.09**	**1.09**	**1.09**	1.01
Average		4.20	8.02	3.81	6.09	4.97	9.31	6.63	3.94	3.79	**3.75**	3.77	**2.65**

Table 5.6: Multi-pass heuristic — *Set 1*: average gaps for each resource weight rule, summary.

		Resource Weight Rule 0 - R0					Resource Weight Rule 1 - R1					Resource Weight Rule 2 - R2					Best min(R0, R1, R2)		Correia et al. (2012)	
		OPN	RPN	min(OPN,RPN)			OPN	RPN	min(OPN,RPN)			OPN	RPN	min(OPN,RPN)						
		Gap (%)	Gap (%)	Gap (%)	CPU (sec)	MCNFP (sec)	Gap (%)	Gap (%)	Gap (%)	CPU (sec)	MCNFP (sec)	Gap (%)	Gap (%)	Gap (%)	CPU (sec)	MCNFP (sec)	Gap (%)	CPU (sec)	Gap (%)	CPU (sec)
SF	1	2.92	2.81	**2.13**	0.413	0.145	2.81	2.63	2.14	0.398	0.157	2.98	2.54	2.14	0.380	0.152	2.10	1.191	23.82	0.477
	0.75	3.72	4.41	3.07	0.344	0.123	3.98	4.06	3.01	0.337	0.118	3.86	4.22	**3.01**	0.330	0.130	2.55	1.010	27.51	0.440
	0.5	5.62	5.66	3.88	0.269	0.089	5.12	4.93	3.49	0.263	0.082	4.80	4.27	**3.13**	0.259	0.080	2.80	0.791	22.69	0.317
	var.	5.59	5.43	3.99	0.324	0.117	5.14	4.79	3.43	0.324	0.121	5.22	4.61	**3.39**	0.306	0.107	3.15	0.953	27.25	0.471
NC	1.5	6.15	6.94	5.16	0.344	0.118	5.93	5.85	4.44	0.327	0.114	5.99	5.60	**4.41**	0.323	0.118	4.19	0.994	29.94	0.603
	1.8	4.52	4.16	**2.97**	0.336	0.117	4.37	4.33	3.16	0.339	0.126	4.36	4.06	3.01	0.316	0.111	2.54	0.992	25.15	0.433
	2.1	2.72	2.62	1.68	0.332	0.121	2.48	2.13	1.46	0.325	0.119	2.30	2.08	**1.33**	0.316	0.124	1.21	0.973	20.87	0.243
MRS	0.1250	7.07	8.07	5.80	0.307	0.103	7.07	7.11	5.48	0.299	0.110	7.02	6.66	**5.30**	0.284	0.107	4.84	0.890	22.81	0.589
	0.1563	3.04	2.10	**2.10**	0.387	0.146	3.04	2.46	2.22	0.388	0.157	3.19	2.22	2.22	0.378	0.153	2.10	1.152	27.25	0.491
	0.1625	5.80	4.26	3.29	0.282	0.076	4.82	3.85	3.12	0.272	0.082	4.31	3.20	**2.54**	0.272	0.088	2.54	0.826	22.08	0.345
	0.1667	4.04	4.89	3.17	0.333	0.126	4.17	4.38	2.87	0.323	0.114	3.95	4.65	**2.84**	0.309	0.110	2.60	0.965	30.17	0.446
	0.1875	1.88	1.41	0.90	0.378	0.130	1.36	1.14	**0.45**	0.363	0.130	1.61	1.08	0.53	0.348	0.129	0.28	1.088	23.40	0.251
	0.2083	2.29	1.83	1.24	0.365	0.137	1.96	1.71	**1.16**	0.369	0.135	1.94	1.71	**1.16**	0.361	0.131	1.01	1.096	28.08	0.266
Average		4.46	4.58	3.27	0.337	0.119	4.26	4.10	3.02	0.330	0.120	4.22	3.91	**2.92**	**0.318**	0.117	**2.65**	**0.986**	25.32	0.426

OPN (RPN): Original (Reversed) Precedence Network; CPU: Total CPU time; MCNFP: Total CPU time spent in solving minimum cost network flow problems.

Set 2

We recall that *Set 2* corresponds to 216 larger instances of the MSRCPSP created using the generator developed in Section 3.4.3 and for which no information is available concerning their optimal values. This is the type of instances for which a good heuristic is of great relevance since hardly an exact approach can be considered (as our results presented next make clear).

We follow the same reasoning that we adopted for *Set 1* and start by analyzing separately the behavior of the different activity priority rules. In addition to the single-priority rules summarized in Table 5.1 we study again the behavior of the three multi-priority rules also considered for *Set 1*, namely $LST+GRPW^*$, $LST + LFT$ and $LST + MTS$. This choice followed a reasoning similar to the one already discussed for the smaller instances.

Table 5.7 displays the number of best solutions achieved by each activity priority rule for each class of test instances. For a given instance, the best solution was obtained by running the selected activity priority rule with the 3 resource weight rules and with the original and reversed precedence networks, choosing at the end the minimum of these 6 makespan values.

The best performing activity priority rule was the multi-priority rule $LST + GRPW^*$, which reached the best upper bounds in 63 instances. The single-priority rule *LST* appears in the second position in this ranking followed by the multi-priority rules $LST + MTS$ and $LST + LFT$. As in the instances in *Set 1*, again, the worst results were obtained with the *SPT* activity priority rule.

These results show some consistency when looked at together with the results obtained for *Set 1*. Accordingly, the heuristic reveals some robustness, which is an important feature for an approximate approach.

The previous analysis does not give an indication about the performance of the heuristic. In order to get such indication, we computed the gaps (in percentage) of both the proposed heuristic and the one developed by Correia et al. (2012) in relation to the length of the critical path as

$$gap = \frac{Z^H - Z^{CP}}{Z^{CP}} \times 100\%,$$

where Z^H denotes the upper bound provided by the heuristic and Z^{CP} denotes the length of the critical path. We note that despite being a typically weak lower bound, the length of the critical path has been widely used to evaluate the performance of approximate methods for the RCPSP (cf. Kolisch, 1996b, Kolisch and Hartmann, 2006).

Regarding the developed multi-pass heuristic, the upper bound considered for each instance was the best solution obtained after running this heuristic with each distinct combination of the 3 resource weight rules with the 11 activity priority rules and with the 2 precedence networks (original and reversed), thus totalizing 66 passes. Additionally, we measured the CPU time required by the general solver CPLEX to reach a feasible solution at least as good as the one provided by the multi-pass heuristic. With this purpose, we implemented the continuous-time model P_{CT} presented in Section 3.2. The upper bound considered for computing the windows for the start times of the activities was the sum of their processing times. A time limit of 36000 seconds (10 hours) of CPU time was set for each instance and all the other CPLEX parameters were used with default settings.

Table 5.8 summarizes the output of these experiments. This table contains 11 columns. Each row is associated with a distinct combination of *SF*, *NC*, and *MRS*, and hence refers to the average results for 6 instances. The results associated with the proposed multi-pass heuristic are displayed in Columns 4, 5 and 6. Column 4 presents the gap; Columns 5 and 6 refer to the average CPU time (in seconds) spent computing the best feasible solution and in solving the minimum cost network flow problems during the computation of such solution, respectively. The gap depicted in Column 4 was computed for each instance considering the makespan retrieved after performing the 66 passes of the proposed multi-pass heuristic (i.e., the best makespan achieved). Analogously to Columns 4 and 5, Columns 7 and 8 present

Table 5.7: Multi-pass heuristic — *Set 2*: performance analysis of the activity priority rules.

SF	*NC*	*MRS*	LFT	LPT	LST	MIS	MTS	SPT	GRPW	GRPW*	LST + GRPW*	LST + LFT	LST + MTS
1	1.5	0.0625	4	1	0	1	2	0	1	1	0	0	1
		0.0781	1	0	3	0	0	0	0	1	1	4	1
		0.0938	1	0	3	0	0	0	2	2	3	3	3
	1.8	0.0625	1	1	0	0	0	0	0	3	2	2	2
		0.0781	0	0	4	1	1	0	0	0	4	1	4
		0.0938	2	0	4	1	1	0	0	0	4	3	4
	2.1	0.0625	0	0	1	1	2	1	1	1	1	0	1
		0.0781	2	1	2	1	0	0	0	1	2	2	2
		0.0938	3	0	1	0	1	0	0	3	1	2	0
0.75	1.5	0.0625	0	0	1	0	1	0	1	2	2	1	1
		0.0792	1	0	0	1	2	0	0	1	2	1	3
		0.0938	0	0	4	0	0	0	0	2	4	2	3
	1.8	0.0625	0	0	1	0	1	0	1	4	1	0	0
		0.0792	1	0	3	0	1	0	0	2	2	1	2
		0.0938	1	0	3	0	2	0	2	1	2	1	2
	2.1	0.0625	2	1	0	3	0	0	1	1	0	0	0
		0.0792	1	0	1	0	0	0	0	3	2	2	2
		0.0938	1	0	4	1	2	0	0	2	4	5	4
0.5	1.5	0.0625	1	0	0	2	2	1	0	1	2	1	2
		0.0781	0	0	2	0	1	0	1	2	2	1	0
		0.0938	1	0	1	0	0	0	0	3	3	3	2
	1.8	0.0625	2	0	0	0	0	0	3	0	2	1	2
		0.0781	2	0	0	1	0	1	1	1	1	1	0
		0.0938	0	0	1	0	2	0	1	4	1	1	1
	2.1	0.0625	1	0	2	3	1	0	1	1	1	2	1
		0.0781	0	1	2	1	1	0	2	0	1	2	1
		0.0938	4	0	3	0	1	0	0	3	1	2	1
var.	1.5	0.0625	2	0	1	0	0	1	0	0	0	2	0
		0.0792	2	0	1	0	1	0	0	2	1	0	0
		0.0938	0	0	4	0	0	0	0	2	1	1	1
	1.8	0.0625	1	1	1	1	1	0	1	0	1	0	1
		0.0792	2	0	2	0	1	0	0	1	3	3	3
		0.0938	2	0	1	2	1	0	0	1	2	2	2
	2.1	0.0625	1	1	1	1	0	1	0	1	1	1	1
		0.0792	2	0	0	0	2	0	0	2	1	0	1
		0.0938	2	0	2	0	2	0	0	0	2	3	3
Average			46	7	59	21	32	5	19	54	**63**	56	57

the gap and the CPU time (in seconds) spent computing a feasible solution obtained by the heuristic developed by Correia et al. (2012), respectively. Column 9 presents the number of instances where the general solver was able to find a solution at least as good as the one provided by the multi-pass heuristic; Column 10 displays the number of instances where an out of memory error occurred. Finally, Column 11 depicts the average CPU time (in seconds) taken by CPLEX to reach solutions at least as good as the ones provided by the proposed heuristic.

From the last row of Table 5.8, we observe that the average gaps achieved by the multi-pass heuristic are roughly 40% smaller than the ones associated with the heuristic developed by Correia et al. (2012). Additionally, we note that the computational effort associated with the former is nearly a third of the required by the latter. A deeper analysis reveals that the multi-pass heuristic obtains the smallest gaps for every combination of *SF*, *NC* and *MRS*. We also observe from the last row of Table 5.8 that in 4 (out of 216) instances, CPLEX failed due to an out of memory error and it was only for 60 instances (out of 216) that CPLEX was able to achieve an upper bound at least as good as the one provided by the multi-pass

heuristic. To reach such upper bound, CPLEX required, on average, 4861 seconds, which represents a significant increase in computational effort.

Table 5.8: Multi-pass heuristic — *Set 2*: multi-pass heuristic and CPLEX results.

			Multi-pass heuristic			Correia et al. (2012)		IBM ILOG CPLEX 12.6		
SF	*NC*	*MRS*	Gap (%)	CPU (sec)	MCNFP (sec)	Gap (%)	CPU (sec)	# SAGH	# OOM	SAGH CPU (sec)
1	1.5	0.0625	104.80	4.718	1.937	137.92	43.798	0	0	
		0.0781	52.85	6.726	3.257	91.51	26.081	0	0	
		0.0938	15.97	9.007	4.182	63.56	7.921	0	0	
	1.8	0.0625	91.01	4.598	1.708	121.03	30.576	0	0	
		0.0781	31.38	6.634	3.317	63.64	17.955	0	0	
		0.0938	15.92	9.009	4.114	66.28	6.339	0	0	
	2.1	0.0625	63.94	4.611	1.834	95.42	17.504	0	0	
		0.0781	22.45	6.632	3.272	55.97	8.242	0	0	
		0.0938	10.44	9.176	4.120	62.22	3.447	0	0	
0.75	1.5	0.0625	109.04	3.356	1.306	133.74	30.480	1	0	14388.300
		0.0792	55.03	4.085	1.516	91.49	21.333	0	0	
		0.0938	29.55	5.213	1.794	79.73	9.363	0	0	
	1.8	0.0625	104.97	3.335	1.240	121.49	16.896	2	0	7614.840
		0.0792	36.61	4.182	1.594	75.28	13.497	0	0	
		0.0938	15.95	5.101	1.800	58.00	5.797	0	0	
	2.1	0.0625	60.78	3.257	1.181	84.55	10.335	3	1	16144.487
		0.0792	19.17	4.005	1.472	56.54	7.129	0	1	
		0.0938	12.58	4.940	1.859	53.77	3.852	2	0	1661.307
0.5	1.5	0.0625	131.36	2.468	0.843	141.02	9.133	5	0	341.513
		0.0781	80.97	2.965	1.047	97.92	13.184	4	0	6481.581
		0.0938	37.17	3.231	1.034	81.05	10.069	1	0	47.609
	1.8	0.0625	105.44	2.398	0.832	109.94	7.667	6	0	562.588
		0.0781	48.71	2.820	0.911	79.54	9.247	4	1	10115.969
		0.0938	31.28	3.489	1.144	67.38	7.775	3	0	3040.920
	2.1	0.0625	92.49	2.510	0.879	105.59	4.971	6	0	558.843
		0.0781	49.16	2.796	0.938	76.85	5.332	6	0	1141.913
		0.0938	9.56	3.085	1.091	40.71	5.188	4	0	5181.150
var.	1.5	0.0625	148.84	3.286	1.201	174.88	19.345	2	0	10451.225
		0.0792	43.91	3.778	1.434	82.04	21.286	0	0	
		0.0938	30.65	4.731	1.670	81.12	13.779	0	0	
	1.8	0.0625	86.39	3.075	1.134	106.10	11.839	3	0	6076.100
		0.0792	34.35	3.820	1.526	72.32	14.533	0	0	
		0.0938	30.13	4.906	1.938	82.03	8.673	0	0	
	2.1	0.0625	56.76	3.080	1.181	76.82	8.117	4	0	9602.805
		0.0792	32.94	3.973	1.487	66.91	7.810	2	1	5642.696
		0.0938	13.70	4.658	1.775	58.13	4.982	2	0	4952.835
Average			53.23	**4.435**	1.766	86.46	12.874	**60**	4	4861.331

CPU: Total CPU time; MCNFP: Total CPU time spent in solving minimum cost network flow problems; SAGH: solutions at least as good as those obtained by the multi-pass heuristic; OOM: out of memory error.

Similarly to the instances in *Set 1*, we investigated the influence of *SF*, *NC* and *MRS* on the quality of the results. With this purpose, we present Tables 5.9 and 5.10. The former was obtained by aggregating the results presented in Table 5.7 according to the different values of *SF*, *NC*, and *MRS*; the latter results from doing the same to Table 5.8.

We recall that for each skill factor value there are 54 instances and that each network complexity value is associated with 72 instances. The modified resource strength of 0.0625 and 0.0938 occurs in 72 instances each whereas the other two values of modified resource strength occur in 36 instances each.

Observing Table 5.9, we notice that (as expected) there is no rule that was the best one for all the

values of *SF*, *NC* and *MRS*. The best performing activity priority rule was the multi-priority rule *LST* + *GRPW**, which obtained the best results in a total of 5 distinct parameter values. This rule produced the highest number of best solutions in 2 out of 4 *SF* values (1 and 0.75) and in 2 out of 3 *NC* values (1.5 and 1.8). As for *MRS*, we observe that the best results were attained by the single-priority rules *LST* (for *MRS* values of 0.0781 and 0.0938) and *GRPW** (for *MRS* values of 0.0625 and 0.0792).

In Table 5.10, we observe that for the 54 instances with *SF* = 1, CPLEX was not able to find any solution at least as good as the one provided by the heuristic within 10 hours of CPU time. The best performance of CPLEX in terms of skill factor occurred in instances with *SF* = 0.5. Regarding the network complexity, the number of feasible solutions found by CPLEX increased as the *NC* value increased. In terms of the modified resource strength, the value 0.0625 originated the best results whereas the value 0.0792 yielded the worst ones. We also observe that the multi-pass heuristic achieves the smallest gaps for every value of *SF*, *NC* and *MRS* in a significantly smaller computational time than the heuristic by Correia et al. (2012). This improvement in the gaps is more noticeable in the results associated with *MRS* = 0.0792 and *MRS* = 0.0938, where the gaps provided by the proposed heuristic are, respectively, 50% and 68% smaller than the ones obtained with the heuristic by Correia et al. (2012).

Table 5.9: Multi-pass heuristic — *Set 2*: performance analysis of the activity priority rules, summary.

		LFT	LPT	LST	MIS	MTS	SPT	GRPW	GRPW*	LST + GRPW*	LST + LFT	LST+ MTS
SF	1	14	3	**18**	5	7	1	4	12	**18**	17	**18**
	0.75	7	1	17	5	9	0	5	18	**19**	13	17
	0.5	11	1	11	7	8	2	9	**15**	14	14	10
	var.	**14**	2	13	4	8	2	1	9	12	12	12
NC	1.5	13	1	20	4	9	2	5	19	**21**	19	17
	1.8	14	2	20	6	11	1	9	17	**25**	16	23
	2.1	19	4	19	11	12	2	5	18	17	**21**	17
MRS	0.0625	**15**	5	8	12	10	4	10	**15**	13	10	12
	0.0781	5	2	**13**	4	3	1	4	5	11	11	8
	0.0792	9	0	7	1	7	0	0	**11**	**11**	7	**11**
	0.0938	17	0	**31**	4	12	0	5	23	28	28	26
Average		46	7	59	21	32	5	19	54	**63**	56	57

Table 5.10: Multi-pass heuristic — *Set 2*: multi-pass heuristic and CPLEX results, summary.

		Multi-pass heuristic			Correia et al. (2012)		IBM ILOG CPLEX 12.6		
		Gap (%)	CPU (sec)	MCNFP (sec)	Gap (%)	CPU (sec)	# SAGH	# OOM	SAGH CPU (sec)
SF	1	45.42	6.790	3.082	84.17	17.985	0	0	
	0.75	49.30	4.164	1.529	83.84	13.187	8	2	10171.757
	0.5	65.13	2.862	0.969	88.89	8.063	39	1	2860.841
	var.	53.07	3.923	1.483	88.93	12.263	13	1	7594.848
NC	1.5	70.01	4.464	1.768	104.66	18.814	13	0	4844.019
	1.8	52.68	4.447	1.772	85.25	12.566	18	1	4801.119
	2.1	37.00	4.394	1.757	69.46	7.242	29	3	4906.465
MRS	0.0625	96.32	3.391	1.273	117.37	17.555	32	1	5125.924
	0.0781	47.59	4.762	2.124	77.57	13.340	14	1	5231.548
	0.0792	37.00	3.974	1.505	74.10	14.265	2	2	5642.696
	0.0938	21.07	5.545	2.210	66.17	7.265	12	0	3593.604
Average		53.23	**4.435**	1.766	86.46	12.874	**60**	4	4861.331

CPU: Total CPU time; MCNFP: Total CPU time spent in solving minimum cost network flow problems; SAGH: solutions at least as good as those obtained by the multi-pass heuristic; OOM: out of memory error.

5.3.2 Biased Random-key Genetic Algorithm — BRKGA

In this section, we report the numerical experiments performed to evaluate the performance of the proposed BRKGA. We first discuss the methodology for selecting the values for the parameters of the developed BRKGA, thus fine-tunning the algorithm. Afterwards, the computational results are reported.

Fine-tuning the BRKGA

The BRKGA introduced in Section 5.2 will become a heuristic for our problem only after the values for its parameters are specified. By checking the literature on genetic algorithms and their applications, we conclude that no method exists for identifying the best values for the parameters of this metaheuristic with respect to some particular problem. Hence, it becomes necessary to conduct a series of preliminary computational tests to find a good set of values for the parameters of the BRKGA.

We performed the preliminary tests using all instances of *Set 1*, since these instances have been intensively studied in the literature and we know the optimal value for most of them. Considering small-sized test beds could compromise the representativeness of the 36 distinct classes of instances defined by the combination of their values of *SF*, *NC*, and *MRS*. Moreover, we assumed that building a hybrid set consisting of instances from *Set 1* and *Set 2* would not allow a greater variability of the values of parameters that define an instance of this problem, since the proportions of these values, which are indexed to the size of the instances, remain roughly constant for both sets of instances. Therefore, we expect the BRKGA to have a similar performance, for a given configuration, on both *Set 1* and *Set 2*. Thus, we assume that a good configuration for *Set 1* will also have a good performance for the larger instances in *Set 2*.

In order to get a hint in terms of good ranges for the values of the parameters of the BRKGA, we referred to Gonçalves et al. (2008), Mendes et al. (2009), and Gonçalves et al. (2011) where a BRKGA framework has been successfully used to solve resource-constrained project scheduling problems.

Unlike the BRKGA frameworks proposed in the aforementioned studies, we consider 3 variants of the BRKGA with regard to the type of decoder employed, namely: (i) all the chromosomes are decoded with the PSS presented in Section 5.1.3; (ii) all the chromosomes are decoded with the SSS proposed in Section 5.2.2; and (iii) the decoder mechanism applied to each chromosome depends on the value

of the allele in the position $n + K + 1$ (see Figure 5.2). These experiments allow us to evaluate if one of the variants performs better than the others. Regarding the precedence network schemes (original or reversed), the computational results presented in Table 5.4 indicate that no scheme was always better than the other. This means that, for some instances, the original precedence network originated better feasible solutions than when the reversed precedence network was applied, while for the remaining instances the best results were achieved by the reversed precedence network. However, it was not possible to identify any class of instances (defined by *SF*, *NC*, *MRS*) where a precedence network scheme performs better than the other. Therefore, we decided to use both network schemes in the chromosomes in order to diversify the search space. We focus our deeper analysis in the aspects that clearly may be decisive for the performance of the new heuristic framework, namely the decoders and the proportions of elite and mutant populations of the proposed BRKGA.

Hence, the precedence network scheme considered for decoding a chromosome in every of the referred variants, depends on the value of the gene $n + K + 2$. Regarding the values of the other BRKGA parameters, Table 5.11 presents the ranges of values considered.

Table 5.11: BRKGA — Preliminary tests for determining the values for the parameters of the BRKGA.

Population size (p)	$5 \times n$ chromosomes
Probability of inheriting an allele from the elite parent (ρ_e)	0.7
Percentage of elite solutions in each generation (p_e)	$\{10, 15\}$
Percentage of mutant solutions in each generation (p_m)	$\{15, 20, 30\}$
Decoder (δ)*	{PSS, SSS, Both}
Stopping criterion ($\mathcal{G}$)	$5 \times n$ generations
Fitness	makespan (smaller is better)

* We considered both original and reversed precedence networks.

A total of 18 distinct configurations are obtained by combining the values in Table 5.11. Each BRKGA configuration was set to run 5 times for each instance (using a distinct seed for the random number generation at the beginning of each run). Accordingly, for each instance tested, it is possible to compute the average and minimum values for the makespan. The runs are independent from each other and terminate when the maximum number of generations is reached. We present in Table 5.12, the average results for every configuration tested. This table consists of two main sets of Columns: (i) Columns 4–8 refer to the results after $n/2$ generations, and (ii) Columns 9–13 to the results after $5 \times n$ generations. The scheduling generation scheme considered as well as the values of p_e and p_m are presented in Columns 1–3, respectively. Each row depicts the average values for the 216 instances in *Set 1*. In terms of percentage gaps, their average and minimum values are indicated in Columns 4 and 9, and Columns 5 and 10, respectively. The average makespan values and their associated standard deviations are presented in Columns 6 and 11, and Columns 7 and 12, respectively. Columns 8 and 13 contain the total CPU time (in seconds) used in the 5 runs until reaching $\frac{n}{2}$ generations and $5 \times n$ generations respectively. Each gap (in percentage) is computed as

$$gap = \frac{Z^{\mathcal{B}} - Z^{LB}}{Z^{LB}} \times 100\%,$$

where $Z^{\mathcal{B}}$ denotes the upper bound provided by the BRKGA and Z^{LB} denotes the optimal value or the best known lower bound obtained by Correia et al. (2012).

The makespan values and gaps presented in Table 5.12 allow us to conclude that, as expected, with $5 \times n$ generations it was possible to obtain better upper bounds than those obtained with $n/2$ generations.

However, this improvement in the quality of the feasible solutions was achieved at the expense of about 10 times more CPU time, which seems not to be compensatory.

Table 5.12: BRKGA — Results of the preliminary tests.

			$\mathcal{G} = \frac{n}{2}$ generations					$\mathcal{G} = 5 \times n$ generations				
δ	p_e (%)	p_m (%)	Gap (%)		Makespan		Total time CPU (sec)	Gap (%)		Makespan		Total time CPU (sec)
			Avg.	Min.	Avg.	Std. dev.		Avg.	Min.	Avg.	Std. dev.	
PSS	10	15	1.19	0.88	52.296	0.163	55.188	0.92	0.75	52.162	0.095	538.735
		20	1.19	0.83	52.298	0.196	55.322	0.87	0.70	52.135	0.112	540.412
		30	1.17	0.86	52.294	0.176	55.558	0.84	0.69	52.114	0.080	543.484
	15	15	1.20	0.87	52.300	0.179	52.668	0.89	0.73	52.141	0.098	510.533
		20	1.18	0.87	52.294	0.161	52.688	0.88	0.72	52.136	0.085	510.715
		30	1.25	0.92	52.331	0.179	52.785	0.87	0.72	52.130	0.085	513.483
SSS	10	15	1.71	1.08	52.531	0.301	76.939	1.21	0.83	52.281	0.191	750.451
		20	1.79	1.23	52.582	0.278	77.054	1.23	0.81	52.302	0.221	753.332
		30	1.82	1.28	52.604	0.274	77.079	1.20	0.85	52.284	0.195	754.394
	15	15	1.68	1.11	52.531	0.278	73.416	1.19	0.83	52.279	0.205	712.074
		20	1.82	1.16	52.600	0.296	73.436	1.16	0.79	52.262	0.193	712.910
		30	1.82	1.24	52.606	0.286	73.477	1.10	0.75	52.239	0.178	714.160
Both	10	15	1.19	0.82	52.298	0.191	63.326	0.92	0.73	52.158	0.109	613.704
		20	1.23	0.86	52.314	0.197	63.668	0.89	0.72	52.144	0.118	617.794
		30	1.21	0.78	52.310	0.225	64.405	0.88	0.74	52.140	0.086	627.068
	15	15	1.20	0.84	52.303	0.186	60.838	0.89	0.73	52.142	0.094	585.492
		20	1.23	0.87	52.317	0.191	60.640	0.86	0.71	52.131	0.097	583.218
		30	1.26	0.90	52.333	0.203	61.265	0.86	0.72	52.124	0.083	592.537

We observe that configurations using either the PSS or both decoders tended to yield the best results either after $n/2$ generations or after $5 \times n$. Due to the prohibitive computational effort associated with considering a large number of generations, we focus on the results obtained after $n/2$ generations.

From the results obtained after $n/2$ generations, we conclude that the use of the PSS decoder in all the chromosomes provided the best results in terms of the average percentage gaps and CPU time. However, the use of both decoders (PSS and SSS) rendered the best minimum gaps for the majority of the combinations of p_e and p_m. Among the 6 combinations of p_e and p_m, we will adopt $p_e = 10$ and $p_m = 30$ since this was the combination that originated the best results both in terms of the average (provided by the PSS decoder) and in terms of the minimum percentage gaps (achieved when both decoders were used).

In the computational results reported in Tables 5.5 and 5.6, we observe that the multi-pass heuristic obtained higher percentage gaps for instances having less resources. In fact, we know that for the instances in *Set 1*, a smaller value of *MRS* corresponds to a smaller number of resources. Since those instances were also the ones that required less computational time, we take advantage of this behavior and consider a population size determined not only by the number of activities (n) but also by the number of resources (K) in each instance. Once again, we used all the instances in *Set 1* and tested $p = 5 \times \lceil \frac{n \times n}{K} \rceil$. By fixing the number of activities (which is 20 for all instances in *Set 1*), such value for p originates larger population sizes for the instances with a smaller number of resources.

Therefore, we present next more intensive tests, considering the two selected configurations:

i) $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 0.7$ and $\delta = PSS$;

ii) $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 0.7$ and $\delta = Both$.

The maximum number of generations will be considered as the stopping criterion and will be set to $n/2$.

BRKGA results

In this section, we report the results obtained with the adopted configurations above-mentioned. We start by presenting and discussing the results obtained for the instances in *Set 1* and afterwards we focus on *Set 2*.

In what follows, we denote the BRKGA with the single decoder PSS as $BRKGA_{PSS}$ and the BRKGA with the two decoders as $BRKGA_{Both}$. We analyze thoroughly the former configuration and then we point out the differences to when the $BRKGA_{Both}$ is used. This way, we also avoid redundancy in terms of the contents presented.

Set 1

In Table 5.13, we present the results for the instances in *Set 1* considering the instances grouped into 36 classes according to their values of *SF*, *NC* and *MRS*. Each row of this table refers to the average results for 6 instances, which together define a class of instances. The percentage gaps were computed as explained before. In Table 5.14, we summarize the same results in order to highlight their most relevant features.

The information presented in Tables 5.13 and 5.14 can be partitioned into 4 parts as follows: (i) Columns 1–3 indicate the characteristics of the instances; (ii) Columns 4–8 contain the results obtained with the $BRKGA_{PSS}$. In particular, Columns 4 and 5 are associated with the average and minimum gaps achieved, Columns 6 and 7 depict the average makespan values and their associated standard deviation, respectively, and Column 8 contains the total CPU time (in seconds) required to perform the 5 runs; (iii) Columns 9–13 contain the same information as (ii) but for $BRKGA_{Both}$; (iv) Columns 14–15 present the percentage gap obtained by the multi-pass heuristic and the corresponding CPU time, respectively.

From Table 5.13, we observe that the average and minimum gaps achieved by the $BRKGA_{PSS}$ improve the results provided by the multi-pass heuristic in 26 and 27 out of the 36 classes of instances, respectively. In terms of average and minimum gaps, the $BRKGA_{PSS}$ achieved the same results as the multi-pass heuristic in 8 classes of instances, 7 of which correspond to a gap of 0.0%. The $BRKGA_{PSS}$ achieved a gap of 0.0% for all instances in 10 and 19 classes of instances regarding average and minimum gaps, respectively.

The 12 classes of instances whose minimum gaps obtained by the $BRKGA_{PSS}$ are highlighted in boldface indicate that the $BRKGA_{PSS}$ was able to find the optimal value of all these 72 instances but the multi-pass heuristic was not.

The higher gaps yielded by the $BRKGA_{PSS}$ are associated with instances with *SF* = var., *NC* = 1.5, *MRS* = 0.1667. The higher values of the standard deviation of the makespan occur more commonly in the classes of instances associated with the smallest value of *MRS*. These correspond to the hardest instances with fewer resources, hence having larger population sizes and thus consuming more computational time.

The last row of Table 5.13 presents the average results across all the 216 instances in *Set 1*. We observe that the gaps provided by the $BRKGA_{PSS}$ improve the ones of the multi-pass heuristic from 2.65% to 1.10% in the case of the average gaps and to 0.79% in terms of the minimum gaps. Hence, regarding the minimum gaps, the values provided by the $BRKGA_{PSS}$ are roughly 3.4 times smaller than the ones achieved by the multi-pass heuristic. In terms of CPU time, the $BRKGA_{PSS}$ requires an average time of one minute while the multi-pass heuristic needs on average one second. In spite of this difference in the computational time, the supremacy of the $BRKGA_{PSS}$ is clear in terms of the quality of the obtained gaps and one minute of average time is still negligible when we look into the complexity of the MSRCPSP.

Similar conclusions can be drawn for the results obtained using the $BRKGA_{Both}$. That version of the BRKGA achieves better results than the multi-pass heuristic for 26 classes out of 36 classes of instances both in terms of average and minimum gaps. The 9 classes of instances whose minimum gaps obtained by the $BRKGA_{Both}$ are highlighted in boldface indicate that the $BRKGA_{Both}$ was able to find the optimal value of all these 54 instances but the multi-pass heuristic was not.

Moreover, we observe the supremacy of the $BRKGA_{Both}$ over the $BRKGA_{PSS}$ regarding average gaps for 5 classes of instances, regarding minimum gaps for also 5 classes of instances and, in terms of both average and minimum gaps for 1 class of instances. Furthermore, the $BRKGA_{Both}$ was able to reach the optimal solutions for all the instances in the following classes of instances (a total of 12 instances): SF = var., $NC = 1.8, MRS = 0.2083$ and $SF = 0.5, NC = 1.5, MRS = 0.1875$, and the $BRKGA_{PSS}$ was not. This behavior may be associated with the incorporation of the SSS decoder which was also the responsible for the $BRKGA_{Both}$ requiring a slightly higher computational time than the $BRKGA_{PSS}$.

From Table 5.14, one can observe that, for the $BRKGA_{PSS}$, as the SF decreases, the gaps also decrease while the associated computational effort increases. This increase may be justified with the fact that instances with a smaller SF have fewer resources and thus originate larger population sizes. For instances having $SF = 0.5$, the $BRKGA_{PSS}$ achieved solutions that correspond to an improvement of 6 and 22.8 times the results of the multi-pass heuristic, for average and minimum percentage gaps, respectively. Despite the instances associated with $SF = 1$ rendering a smaller improvement, the minimum gaps provided by the $BRKGA_{PSS}$ for these instances are roughly half of the ones obtained when using the multi-pass heuristic.

Looking into the results from the perspective of the network complexity (NC) values, we conclude that an increase in this parameter leads, as expected, to a reduction in the gaps of the BRKGA. The $BRKGA_{PSS}$ was able to reduce the minimum gaps of the instances having $NC = 2.1$ to nearly 0.0%.

The instances having $MRS = 0.1625$ were the ones where the $BRKGA_{PSS}$ produced the smallest gaps with values of 0.16% and 0% for average and minimum gaps, respectively. We point out that all these instances have $SF = 0.5$ and thus correspond to cases that were among the hardest to tackle by the multi-pass heuristic. In this subset of "harder" instances, we also find the ones associated with small values of MRS, such as $MRS = 0.1250$, where the $BRKGA_{PSS}$ originates a minimum gap of 0.83%. This correspond to a major improvement since the multi-pass heuristic produced gaps roughly 6 times higher.

The $BRKGA_{Both}$ improve the results obtained by the $BRKGA_{PSS}$ for $MRS = 0.1563$ and $MRS = 0.1875$, regarding average gaps and for $SF = 1$, $MRS = 0.1875$ and $MRS = 0.2083$ in terms of minimum gaps.

Table 5.13: BRKGA — *Set 1*: comparative analysis.

			BRKGA 5 runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil, p_e = 10\%, p_m = 30\%, \rho_e = 70\%, \mathcal{G} = \frac{n}{2}$										Multi-pass heuristic	
			$\delta = PSS$					$\delta = Both$						
			Gap (%)		Makespan		Total time CPU (sec)	Gap (%)		Makespan		Total time CPU (sec)	Gap (%)	Total time CPU (sec)
SF	*NC*	*MRS*	Avg.	Min.	Avg.	Std. dev.		Avg.	Min.	Avg.	Std. dev.			
1	1.5	0.1250	6.33	6.13	52.433	0.149	61.027	6.47	5.83	52.500	0.448	66.355	9.47	1.096
		0.1563	2.89	2.89	49.167	0.000	55.743	2.97	2.89	49.200	0.075	63.376	4.37	1.145
		0.1875	0.00	0.00	48.667	0.000	56.852	0.00	0.00	48.667	0.000	67.529	0.00	1.374
	1.8	0.1250	0.73	**0.00**	53.033	0.289	59.165	0.77	**0.00**	53.067	0.322	64.993	1.57	1.051
		0.1563	1.19	0.79	47.667	0.149	54.801	1.03	0.79	47.600	0.091	64.405	1.92	1.195
		0.1875	0.46	0.46	45.833	0.000	54.293	0.46	0.46	45.833	0.000	66.847	0.46	1.333
	2.1	0.1250	0.00	**0.00**	64.167	0.000	60.835	0.00	**0.00**	64.167	0.000	69.524	1.07	1.103
		0.1563	0.00	0.00	53.833	0.000	52.455	0.00	0.00	53.833	0.000	61.553	0.00	1.116
		0.1875	0.00	0.00	56.667	0.000	54.155	0.00	0.00	56.667	0.000	67.839	0.00	1.304
0.75	1.5	0.1250	0.70	0.53	59.767	0.166	69.142	0.85	0.26	59.833	0.332	75.654	3.98	0.976
		0.1667	0.33	**0.00**	44.133	0.166	57.961	0.44	**0.00**	44.200	0.231	65.579	0.90	1.000
		0.2083	2.68	2.39	51.433	0.091	54.167	3.06	2.87	51.567	0.091	63.849	3.84	1.145
	1.8	0.1250	2.04	0.82	60.433	0.610	66.369	2.27	1.31	60.567	0.498	71.881	6.47	0.916
		0.1667	2.34	1.72	47.800	0.224	54.704	2.46	2.04	47.833	0.183	63.326	2.93	0.952
		0.2083	0.00	0.00	49.833	0.000	52.925	0.00	0.00	49.833	0.000	64.306	0.00	1.117
	2.1	0.1250	0.95	**0.00**	59.600	0.428	65.550	0.74	0.22	59.533	0.298	70.397	3.99	0.867
		0.1667	0.00	0.00	43.167	0.000	56.201	0.24	0.00	43.267	0.091	63.110	0.00	0.984
		0.2083	0.00	**0.00**	45.000	0.000	52.240	0.00	**0.00**	45.000	0.000	63.240	0.79	1.137
0.5	1.5	0.1250	1.79	0.26	61.633	0.833	81.355	1.96	0.80	61.733	0.584	85.962	6.17	0.749
		0.1625	0.20	**0.00**	49.933	0.224	67.694	0.34	**0.00**	50.000	0.183	73.764	4.27	0.796
		0.1875	0.51	0.51	41.667	0.000	64.173	0.10	**0.00**	41.533	0.075	72.743	0.88	0.890
	1.8	0.1250	0.79	0.34	60.267	0.240	72.780	0.97	0.63	60.367	0.257	77.886	5.19	0.658
		0.1625	0.29	**0.00**	55.467	0.240	70.753	0.41	**0.00**	55.533	0.224	75.775	2.42	0.872
		0.1875	0.14	**0.00**	47.900	0.091	62.440	0.22	**0.00**	47.933	0.091	70.642	0.36	0.846
	2.1	0.1250	0.42	**0.00**	71.967	0.447	78.988	0.39	0.24	71.967	0.183	81.704	5.00	0.713
		0.1625	0.00	**0.00**	55.167	0.000	65.061	0.00	**0.00**	55.167	0.000	73.334	0.93	0.809
		0.1875	0.00	0.00	56.000	0.000	57.938	0.00	0.00	56.000	0.000	66.694	0.00	0.783
var.	1.5	0.1250	3.12	1.58	52.467	0.611	60.477	3.45	2.19	52.633	0.743	68.199	7.55	0.765
		0.1667	6.75	6.56	47.233	0.091	53.469	6.90	6.56	47.300	0.183	61.253	7.41	0.939
		0.2083	0.49	0.49	37.667	0.000	49.560	0.49	0.49	37.667	0.000	60.592	1.42	1.049
	1.8	0.1250	0.91	**0.00**	54.333	0.348	63.006	1.47	0.93	54.633	0.332	69.311	5.16	0.872
		0.1667	2.08	2.08	45.667	0.000	55.702	2.21	1.67	45.733	0.240	62.642	4.02	1.002
		0.2083	0.71	0.51	43.733	0.091	49.464	0.99	0.00	43.833	0.240	59.584	0.00	1.093
	2.1	0.1250	0.38	0.32	63.700	0.075	66.694	0.32	0.32	63.667	0.000	73.867	2.46	0.916
		0.1667	0.52	**0.00**	48.900	0.166	54.834	0.67	0.41	48.967	0.075	63.657	0.33	0.911
		0.2083	0.00	0.00	54.667	0.000	47.308	0.00	0.00	54.667	0.000	60.410	0.00	1.033
Average			1.10	0.79	52.250	0.159	60.008	1.19	0.86	52.292	0.169	68.105	2.65	0.986

Table 5.14: BRKGA — *Set 1*: comparative analysis, summary.

		BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil, p_e = 10\%, p_m = 30\%, \rho_e = 70\%, \mathcal{G} = \frac{n}{2}$										Multi-pass heuristic	
		$\delta = PSS$					$\delta = Both$						
		Gap (%)		Makespan		Total time CPU (sec)	Gap (%)		Makespan		Total time CPU (sec)	Gap (%)	Total time CPU (sec)
		Avg.	Min.	Avg.	Std. dev.		Avg.	Min.	Avg.	Std. dev.			
SF	1	1.29	1.14	52.385	0.065	56.592	1.30	1.11	52.393	0.104	65.824	2.10	1.191
	0.75	1.00	0.61	51.241	0.187	58.807	1.12	0.74	51.293	0.191	66.816	2.55	1.010
	0.5	0.46	0.12	55.556	0.231	69.020	0.49	0.19	55.581	0.177	75.389	2.80	0.791
	var.	1.66	1.28	49.819	0.154	55.613	1.83	1.40	49.900	0.201	64.391	3.15	0.953
NC	1.5	2.15	1.78	49.683	0.194	60.968	2.25	1.82	49.736	0.245	68.738	4.19	0.994
	1.8	0.97	0.56	50.997	0.190	59.700	1.11	0.65	51.064	0.206	67.633	2.54	0.992
	2.1	0.19	0.03	56.069	0.093	59.355	0.20	0.10	56.075	0.054	67.944	1.21	0.973
MRS	0.1250	1.51	0.83	59.483	0.350	67.116	1.64	1.06	59.556	0.333	72.978	4.84	0.890
	0.1563	1.36	1.23	50.222	0.050	54.333	1.33	1.23	50.211	0.055	63.111	2.10	1.152
	0.1625	0.16	0.00	53.522	0.155	67.836	0.25	0.00	53.567	0.135	74.291	2.54	0.826
	0.1667	2.00	1.73	46.150	0.108	55.479	2.15	1.78	46.217	0.167	63.261	2.60	0.965
	0.1875	0.19	0.16	49.456	0.015	58.308	0.13	0.08	49.439	0.028	68.716	0.28	1.088
	0.2083	0.65	0.57	47.056	0.030	50.944	0.76	0.56	47.094	0.055	61.997	1.01	1.096
Average		1.10	0.79	52.250	0.159	60.008	1.19	0.86	52.292	0.169	68.105	2.65	0.986

Set 2

We recall that *Set 2* contains the 216 larger instances and also that by using the multi-pass heuristic it was possible to find the optimal value for 5 of these instances and, of course, an upper bound on that value for the remaining ones. This constitutes all the available information that can be used for evaluating the results provided by the $BRKGA_{PSS}$ and by the $BRKGA_{Both}$. The results presented in Table 5.8 showed that, after 10 hours of CPU time, a general solver was unable to find a solution with the same quality of the one provided by the multi-pass heuristic for 156 instances. For the remaining 60 instances, the solver required, on average, 4861 seconds of CPU time to find a solution of at least the same objective value of the one provided by the multi-pass heuristic. It is for this type of instances, where the use of exact methods becomes impractical, that the development of efficient approximate methods is of particular interest.

In order to evaluate the results provided by both the $BRKGA_{PSS}$ and the $BRKGA_{Both}$ and due to the fact that no information regarding the optimal solutions is available for the majority of these instances, we introduce a new concept referred to as *Performance Ratio* (*PR*). This is a relative ratio that we compute both for the makespan values (PR_m) and for the gaps (PR_g) as follows:

$$PR_m = \frac{Z^{\mathcal{B}^*} - Z^H}{Z^H} \times 100\%, \quad \text{regarding makespan values,}$$

$$PR_g = \frac{D^{\mathcal{B}^*} - D^H}{D^H} \times 100\%, \quad \text{regarding gap values,}$$

where $Z^{\mathcal{B}^*}$ ($D^{\mathcal{B}^*}$) denotes the best upper bound (minimum gap) provided by the $BRKGA_{PSS}$ or by the $BRKGA_{Both}$ and Z^H (D^H) denotes the upper bound (gap) obtained by the multi-pass heuristic. The lower bound considered for computing the percentage gaps for each instance was the length of the corresponding critical path. We illustrate this concept next.

Suppose that for some instance, and considering the $BRKGA_{PSS}$, we obtain $D^{\mathcal{B}^*} = 8\%$ and $D^H = 10\%$. In this case, we have $PR_g = -20\%$, which indicates that the $BRKGA_{PSS}$ provides a gap 20% lower than the multi-pass heuristic.

The analysis of improvements from the makespan point of view (PR_m) is also of great interest, in particular when no lower bound besides the critical path length is available.

For fixed values of *SF* and *NC*, a decrease of the value of *MRS* usually leads to a deterioration of the objective value (i.e., leads to its increase) which consequently results in a larger distance between the optimal value and the critical path length value. It is for this type of instances, particularly when the best known lower bound is most likely poor, that the use of a makespan performance measure is of major relevance.

Table 5.15 presents the detailed results for each class of instances induced by a distinct combination of *SF*, *NC* and *MRS*. Each row of this table (class of instances) presents average results for 6 instances. The results are then summarized in Table 5.16.

The information presented in Tables 5.15 and 5.16 can be partitioned into 4 parts as follows: (i) Columns 1–4 contain the characteristics of each class of instances; (ii) Columns 5–9 are associated with the results obtained by the $BRKGA_{PSS}$. In particular, columns 5 and 6 show the results associated with the *PR* values in terms of makespan (PR_m) and gap (PR_g), respectively and columns 7–9 present the average makespan, standard deviation and total CPU time required by the 5 runs, respectively; (iii) Columns 10–14 refer to the results provided by the $BRKGA_{Both}$ and follow the structure of (ii); (iv) Column 15 depicts the total time required by the multi-pass heuristic.

We note that the $BRKGA_{PSS}$ found the optimal solutions for all the 5 instances for which the multi-pass heuristic identified their optimal value. We did not take into account these instances for computing the values presented in Tables 5.15 and 5.16. Therefore, we indicate, in (the untitled) column 4, inside parentheses, the number of instances not considered for the results presented.

From Table 5.15, we observe a general improvement in terms of both PR_m and PR_g. The $BRKGA_{PSS}$ performed better than the multi-pass heuristic in all the 36 classes of instances. The best results in terms of PR_m were generally attained for the instances associated with smaller values of *MRS*, with a particular emphasis to those having $SF = 0.5$ and $MRS = 0.0625$. In fact, the class of instances that reported the highest improvement regarding PR_m (–13.86%) is actually defined by $SF = 0.5$, $NC = 1.5$ and $MRS = 0.0625$. This class was also the one associated with the highest standard deviation of the average makespan value, which indicates an increased difficulty faced by the $BRKGA_{PSS}$ in attaining solutions of the same fitness consistently for these instances. Analyzing the PR_g values, these seem to follow a different direction. This result may be related to the fact that smaller makespan values are associated with instances with higher values of *MRS* (thus allowing a greater degree of parallelization) for fixed values of *SF* and *NC*. In fact, considering $SF = 0.5$, the class of instances with $NC = 2.1$ and $MRS = 0.0938$ is the one associated with the smallest improvement in the makespan (PR_m = –2.30%) and the one having the greatest gap improvement (PR_g = –35.18%), hence validating our reasoning of evaluating the results of the selected BRKGA configurations in terms of gap and makespan improvement from the results of the multi-pass heuristic.

The $BRKGA_{Both}$ improved the results of the $BRKGA_{PSS}$ for 25 out of the 36 classes of instances. More precisely, the $BRKGA_{Both}$ was better in 5 classes of instances having $SF = 1$, 8 classes of instances with $SF = 0.75$ and 6 classes of instances for both SF = var. and $SF = 0.5$. The best results attained by the $BRKGA_{Both}$ are associated with PR_m = –15.88% and PR_g = –36.57% and have been found in the same class of instances where the $BRKGA_{PSS}$ performed best. A closer look at Table 5.15 allows the identification of a pattern. In fact, as the *SF* decreases, the quality of the results achieved by the $BRKGA_{PSS}$ and by the $BRKGA_{Both}$ in terms of PR_g and PR_m are generally improved.

From Table 5.16 we observe a global improvement with respect to the solutions obtained by the multi-pass heuristic. The quality of the results achieved by the $BRKGA_{PSS}$ tends to increase as the values of *SF* become smaller. Smaller values of *SF* lead to a smaller computational effort. Instances having $SF = 0.5$ are associated with higher values of standard deviation of makespan, a similar behavior to what was observed for *Set 1*. These instances can, in fact, be looked at as "difficult" to tackle. Moreover, it is for these instances that the best results regarding both PR_m and PR_g were achieved.

From the perspective of the network complexity (*NC*), an increase in its value leads to a slight decrease in terms of the CPU time consumed. We note that 3 (out of the 5) instances where the multi-pass heuristic reached the optimal solution, are associated with $NC = 2.1$. Instances having higher values of *NC* usually tend to be easier because a smaller number of activities is likely to be processed in parallel due to the increased number of precedence relations, for a fixed number of activities. Among the different *NC* values, the largest improvements in terms of both PR_m and PR_g were generally achieved for the instances having $NC = 1.8$.

Regarding the *MRS*, the best results in terms of PR_m and PR_g are associated with $MRS = 0.0625$ and $MRS = 0.0938$, respectively. This may be explained by the fact that the instances with higher values of *MRS* are the ones associated with smaller values of average makespan and, as discussed before, these instances typically allow a greater degree of parallelization and hence their corresponding optimal values can be closer to the value of the associated critical path length.

We noticed that the 5 instances for which the multi-pass heuristic attain their optimal solutions, belong to the set of instances having the most resources, which are associated with the highest value of *MRS* considered (0.0938). We observe average values for the 216 instances in this set of PR_m = –5.69% and PR_g = –21.10% for the $BRKGA_{PSS}$. This represents a significant improvement of the results obtained by the multi-pass heuristic.

It is worth noticing that the standard deviation values are larger for the instances having smaller values of either *SF* or *NC* or *MRS*. In terms of CPU time, we should recall that the instances in this set have roughly the double of resources and activities as well as a population and a stopping criterion with

twice the size of the ones considered in *Set 1*.

The aggregated view of Table 5.16 provides strong evidence that the $BRKGA_{Both}$ outperforms the $BRKGA_{PSS}$ for each and every value of *SF*, *NC* and *MRS* at the expense of an increase of 9% on the CPU time which, nonetheless is still negligible given the high complexity of the MSRCPSP. The $BRKGA_{Both}$ achieved the best average results for *Set 2* by obtaining PR_m = –6.05% and PR_g = –22.25%, and hence it may be more appropriate for dealing with instances of large dimensions. In fact, the use of the two decoders may have contributed to a better exploration of the search space of feasible solutions, which is larger for the instances in *Set 2*.

Table 5.15: BRKGA — *Set 2*: comparative analysis.

				BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil, p_e = 10\%, p_m = 30\%, \rho_e = 70\%, \mathcal{G} = \frac{n}{2}$									Multi-pass heuristic
				$\delta = PSS$					$\delta = Both$					
				Performance Ratio (%)		Makespan		Total time CPU (sec)	Performance Ratio (%)		Makespan		Total time CPU (sec)	Total time CPU (sec)
SF	*NC*	*MRS*		PR_m	PR_g	Avg.	Std. dev.		PR_m	PR_g	Avg.	Std. dev.		
1	1.5	0.0625		–5.92	–11.79	97.900	0.783	1158.783	–5.74	–11.30	98.167	0.859	1236.057	4.718
		0.0781		–4.36	–12.77	79.300	0.847	1478.504	–4.13	–12.06	79.300	0.789	1620.044	6.726
		0.0938	(1)	–3.06	–18.70	70.480	0.687	1791.673	–4.16	–26.96	70.200	0.837	1890.474	9.007
	1.8	0.0625		–6.49	–14.61	100.533	1.027	1136.121	–5.91	–13.31	100.800	0.704	1220.869	4.598
		0.0781		–5.17	–27.61	82.400	0.827	1455.465	–5.17	–26.56	82.067	0.763	1582.846	6.634
		0.0938	(1)	–2.41	–33.12	65.160	0.597	1776.367	–2.70	–34.66	65.040	0.649	1888.868	9.009
	2.1	0.0625		–5.33	–14.10	100.000	0.767	1132.525	–5.48	–14.29	99.867	0.753	1211.125	4.611
		0.0781		–3.02	–19.39	85.900	0.562	1451.702	–3.03	–20.18	86.267	0.718	1566.615	6.632
		0.0938		–1.68	–26.39	67.733	0.456	1726.694	–1.93	–29.17	67.633	0.582	1846.476	9.176
0.75	1.5	0.0625		–6.20	–12.67	110.300	0.656	1119.035	–6.57	–13.23	109.700	0.743	1243.254	3.356
		0.0792		–4.75	–14.07	81.967	0.865	1102.655	–5.29	–16.08	81.933	0.965	1192.030	4.085
		0.0938		–3.64	–16.33	69.800	0.611	1135.340	–4.04	–18.13	69.767	0.711	1246.575	5.213
	1.8	0.0625		–8.54	–16.92	110.733	1.067	1142.319	–9.75	–19.18	109.800	1.258	1282.192	3.335
		0.0792		–4.83	–19.95	83.767	0.959	1092.978	–5.25	–22.06	83.700	1.046	1183.419	4.182
		0.0938		–3.52	–27.59	70.800	0.872	1115.094	–3.24	–22.78	70.667	0.659	1226.932	5.101
	2.1	0.0625		–7.73	–20.69	108.967	1.108	1076.178	–8.20	–21.78	108.700	1.332	1181.252	3.257
		0.0792		–3.32	–19.42	77.400	0.673	1045.309	–3.94	–24.91	77.400	0.942	1138.075	4.005
		0.0938	(2)	–1.44	–18.13	71.150	0.537	1109.151	–1.80	–20.63	71.000	0.649	1218.183	4.940
0.5	1.5	0.0625		–13.86	–25.82	120.767	2.408	1139.316	–15.88	–29.44	117.400	2.060	1231.441	2.468
		0.0781		–9.22	–20.41	98.433	1.393	1127.840	–10.34	–23.92	97.067	1.285	1217.346	2.965
		0.0938		–3.62	–24.49	81.500	0.829	1100.186	–3.44	–23.73	81.600	0.810	1170.960	3.231
	1.8	0.0625		–12.77	–25.88	129.200	1.712	1087.627	–12.46	–25.24	129.267	1.748	1179.768	2.398
		0.0781		–8.58	–27.97	88.167	1.193	1050.740	–8.23	–26.56	88.600	1.236	1124.984	2.820
		0.0938		–6.85	–29.21	77.567	0.939	1134.217	–7.10	–30.26	77.600	0.896	1224.682	3.489
	2.1	0.0625		–10.60	–22.31	120.933	1.628	1109.013	–12.16	–25.58	119.467	2.129	1179.548	2.510
		0.0781		–6.58	–22.85	87.100	0.716	1043.272	–7.88	–27.31	86.900	1.406	1112.957	2.796
		0.0938		–2.30	–35.18	75.400	0.316	1027.204	–2.52	–36.57	75.300	0.300	1108.814	3.085
var.	1.5	0.0625		–7.68	–13.21	110.367	1.572	1116.957	–9.34	–15.94	108.167	1.479	1351.140	3.286
		0.0792		–6.23	–24.12	79.533	1.077	1015.154	–5.86	–22.76	79.533	1.009	1120.748	3.778
		0.0938		–2.99	–30.28	70.467	0.745	1061.472	–3.19	–33.63	70.467	0.781	1176.741	4.731
	1.8	0.0625		–9.57	–21.46	111.067	1.703	1033.412	–10.64	–23.63	109.600	1.628	1230.622	3.075
		0.0792		–3.40	–14.15	77.967	0.719	1017.991	–3.78	–17.06	77.867	0.820	1113.140	3.820
		0.0938		–4.24	–19.31	71.400	0.787	1068.803	–3.76	–17.69	71.567	0.715	1181.989	4.906
	2.1	0.0625		–6.37	–17.71	109.433	1.290	997.032	–6.95	–19.13	108.733	1.254	1157.324	3.080
		0.0792		–3.08	–20.06	88.467	0.541	1052.956	–3.11	–20.73	88.433	0.732	1164.139	3.973
		0.0938	(1)	–2.74	–21.49	73.720	0.888	1018.186	–1.93	–15.64	74.000	0.578	1144.390	4.658
Average			(5)	–5.69	–21.10	89.493	0.962	1169.095	–6.05	–22.25	89.148	1.003	1277.896	4.435

Table 5.16: BRKGA — *Set 2*: comparative analysis, summary.

			BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil, p_e = 10, p_m = 30, \rho_e = 70, \mathcal{G} = \frac{n}{2}$										Multi-pass heuristic
			$\delta = PSS$					$\delta = Both$					
			Performance Ratio (%)		Makespan		Total time CPU (sec)	Performance Ratio (%)		Makespan		Total time CPU (sec)	Total time CPU (sec)
			PR_m	PR_g	Avg.	Std. dev.		PR_m	PR_g	Avg.	Std. dev.		
SF	1	(2)	–4.22	–19.60	83.862	0.731	1443.826	–4.28	–20.56	83.862	0.739	1550.017	6.790
	0.75	(2)	–5.02	–18.43	87.827	0.827	1104.039	–5.48	–19.83	87.577	0.933	1212.213	4.164
	0.5		–8.26	–26.01	97.674	1.237	1091.046	–8.89	–27.62	97.022	1.319	1172.278	2.862
	var.	(1)	–5.19	–20.17	88.317	1.038	1042.898	–5.46	–20.78	87.853	1.008	1182.962	3.923
NC	1.5	(1)	–6.00	–18.72	89.499	1.044	1187.180	–6.53	–20.51	88.868	1.030	1299.865	4.464
	1.8	(1)	–6.42	–23.01	89.400	1.040	1167.471	–6.55	–23.09	89.217	1.015	1278.211	4.447
	2.1	(3)	–4.63	–21.57	89.583	0.796	1152.157	–5.04	–23.17	89.365	0.962	1254.966	4.394
MRS	0.0625		–8.42	–18.10	110.850	1.310	1104.026	–9.09	–19.34	109.972	1.329	1225.383	3.391
	0.0781		–6.15	–21.83	86.883	0.923	1267.920	–6.46	–22.77	86.700	1.033	1370.799	4.762
	0.0792		–4.27	–18.63	81.517	0.806	1054.507	–4.54	–20.60	81.478	0.919	1151.925	3.974
	0.0938	(5)	–3.28	–25.25	72.230	0.692	1247.489	–3.38	–25.98	72.206	0.681	1352.096	5.545
Average			–5.69	–21.10	89.493	0.962	1169.095	–6.05	–22.25	89.148	1.003	1277.896	4.435

5.3.3 Mathematical models and lower bounds

This section focuses on reporting the numerical results associated with the models and lower bounds presented in Chapter 4. Since the formulations that we present require an upper bound for each instance (to compute the latest start times of the activities), we adopt the best upper bound (minimum value) among those provided by the multi-pass heuristic, by the $BRKGA_{PSS}$ and by the $BRKGA_{Both}$. We begin by discussing the results associated with the mathematical formulations and then we focus on evaluating the quality of the lower bounds provided by the *SbCB* and by the *SbCSLB*.

Mathematical models

The value of M used in model P_{CT} was set to the best available upper bound. As discussed in the previous chapter, the objective value of a linear programming relaxation of a given mathematical formulation for this problem, constitutes a lower bound on its optimal value. Such value can be used both to compute gaps of approximate algorithms more accurately and as a lower cutoff, for instance within optimization models. The quality of the linear programming relaxation of a given model can be assessed by computing, for each instance, its corresponding LP-relaxation gap (*LP* gap) as follows:

$$gap = \frac{Z^{UB} - \lceil Z^{LP} \rceil}{Z^{UB}} \times 100\%,$$

with Z^{UB} denoting the best available upper bound and Z^{LP} denoting the optimal value of the corresponding linear programming relaxation. The best available upper bound for each instance is the minimum among all the upper bounds provided by the heuristics discussed previously and by the mathematical models considered in this computational study.

We start by presenting and analyzing the results for the 216 instances in *Set 1* and then we move to the 216 larger instances in *Set 2*.

Set 1

After performing some preliminary experiments, it was possible to determine the mathematical model that performed best, i.e., the model that achieved the highest number of optimal solutions (for instances in *Set 1*). Then, it seemed natural to assess whether supplying information of a lower bound (provided by other source than its corresponding linear programming relaxation) had any impact on either the computational time required to prove optimality or the number of optimal solutions achieved by such model. Instead of providing a lower cutoff on the objective function of that model, we opted to set the earliest start time of the dummy activity $n + 1$ to a specific lower bound value. More specifically, we considered the following:

1. $\max_{zLP}$: the maximum LP-relaxation value across all mathematical formulations;
2. $\max_{zLB}$: the maximum among the values derived by the two lower bound methods presented in Section 4.3 and $\max_{zLP}$.

Table 5.17 summarizes the results obtained for instances in *Set 1*. This table contains 8 columns. The first column contains the models considered. More specifically, $'X + (A)'$ denotes the addition of the inequalities (A) to model X, $'(RT)'$ identifies the application of the reduction test presented in Section 4.1, and $'X + (\text{A} - \text{B})'$ denotes the addition of all inequalities associated with a label between (A) and (B), inclusive, to model X. Column 2 reports the number of instances (out of 216) that were solved to optimality (# opt) within the given CPU time of 18000 seconds and Column 3 indicates the average time (in CPU seconds) required to solve such instances. Column 4 depicts the number of instances where the optimality of the best feasible solution found was not proved within the stipulated time limit (#tl). For those instances, the average MIP gap (%) reported at CPLEX termination is shown in Column

5. Column headed by "# inf" indicates the number of instances for which no feasible solution was found within the predefined CPU time limit. Column 7 indicates the relative percentage deviation between the objective value of the best known feasible solution and the linear programming relaxation bound (Z^{LP}) of the corresponding model. Finally, Column 8 presents the average time (also in seconds) required by the linear programming relaxations.

From this table, we can observe that the model that performed best in terms of solving the problem to optimality was among the continuous-time models derived from P_{CT}. In fact, besides being able to find the most number of optimal solutions (when compared to the discrete-time formulations), these models were also the ones that required the smallest average CPU time for proving their optimality. The smallest value in the column titled "# inf" was also obtained by these formulations. Moreover, we observe that the progressive inclusion of both the valid inequalities and the reduction test into model P_{CT} contributes to a higher number of instances where a feasible solution is found. In fact, model P_{CT} fails to find a feasible solution within the time limit of 18000 seconds for 4 instances, but when all the valid inequalities and reduction test are applied, such number reduces to 0. Furthermore, we find that the addition of the valid inequalities (see Section 4.1.2) and the reduction test (see Section 4.1.1) contributed to increase the value of the linear programming relaxation for some instances. Model P_M and model P_M + (4.39 + 4.40) render the worst results in terms of the number of optimal solutions found and the number of instances for which no feasible solution was obtained, respectively. In general, the results obtained by the discrete-time models with the disaggregate precedence constraints outperformed the results provided by the corresponding models without those constraints. Among these models, model P_{DDT} achieved more optimal solutions and required a lower CPU time than the other discrete-time models. Still in Table 5.17, we can observe that the linear programming relaxations of the continuous-time models were the fastest but produced some of the worst lower bounds. Being the computation of feasible solutions to the MSRCPSP much more difficult than to the classical resource-constrained project scheduling problem, it is not surprising the weak CPLEX's ability to compute optimal solutions for the former problem. Models P_M and P_{MDDT} were the most time consuming and were not able to produce better lower bounds than the other discrete-time models. Models P_{DT} and P_N produce the same lower bounds. Such behavior is also observed for models P_{DDT} and P_{NDDT}.

Table 5.17: Mathematical formulations — *Set 1*: overall results.

Model	# opt	CPU (sec)	# tl	MIP gap (%)	# inf	*LP* gap (%)	*LP* CPU (sec)
P_{CT}	181	299.455	31	6.68	4	7.24	0.063
P_{CT} + (4.7)	184	352.530	29	6.88	3	7.24	0.104
P_{CT} + (4.7) + (*RT*) + (4.1 – 4.5)	207	137.021	9	5.43	0	6.89	0.479
P_{CT} + (4.7) + (*RT*) + (4.1 – 4.5) + max$_{LP}$	205	56.928	11	6.80	0	—	—
P_{CT} + (4.7) + (*RT*) + (4.1 – 4.5) + max$_{LB}$	205	53.267	11	6.86	0	—	—
P_{DT}	143	479.575	28	6.32	45	6.86	2.049
P_{DT} + (4.38)	144	459.962	29	7.92	43	6.86	5.428
P_{DDT}	151	395.942	31	5.83	34	6.01	5.540
P_{DDT} + (4.38)	148	782.641	26	5.70	42	6.01	10.324
P_M	138	503.199	25	8.53	53	7.17	19.279
P_M + (4.39 + 4.40)	138	543.596	23	6.53	55	7.17	27.702
P_{MDDT}	148	623.032	30	6.21	38	6.01	15.596
P_{MDDT} + (4.39 + 4.40)	152	660.396	29	6.05	35	6.01	22.856
P_N	150	859.261	34	8.55	32	6.86	3.142
P_N + (4.39 + 4.40)	142	532.324	36	7.02	38	6.86	5.639
P_{NDDT}	150	653.170	32	7.05	34	6.01	5.094
P_{NDDT} + (4.39 + 4.40)	150	938.423	28	6.04	38	6.01	10.193

In order to highlight the main differences in the results provided by both the continuous-time model P_{CT} and a discrete-time model, we present next the results grouped according to *NC*, *SF*, and *MRS*. The main reason from selecting the standard continuous-time model P_{CT} for this comparison is associated with the fact that the other continuous-time models include information (e.g., valid inequalities, reduction tests, lower bound values) that is not available to any discrete-time formulation. Regarding the discrete-time model, we decided to analyze P_{NDDT} in more depth since it was the fastest to solve the linear programming relaxation among the models that produced an average linear programming relaxation gap of 6.01%.

Table 5.18 concerns model P_{CT}. The first two columns of this table contain information about the instances and the third column "# inst" contains the number of instances associated with each row of the table. The remaining columns have the same meaning of the corresponding columns in Table 5.17.

Table 5.18: Model P_{CT} — *Set 1*: results per each value of *SF*, *NC* and *MRS*.

		# inst	# opt	CPU (sec)	# tl	MIP gap (%)	# inf	*LP* gap (%)	*LP* CPU (sec)
SF	1	54	39	265.795	11	10.45	4	4.60	0.159
	0.75	54	43	393.483	11	4.69	0	6.34	0.038
	0.5	54	53	11.213	1	2.08	0	10.88	0.011
	var.	54	46	572.203	8	4.82	0	7.16	0.041
NC	1.5	72	52	537.767	17	6.94	3	8.27	0.099
	1.8	72	63	289.244	8	7.02	1	6.62	0.065
	2.1	72	66	121.422	6	5.51	0	6.84	0.023
MRS	0.1250	72	56	437.042	15	9.04	1	15.62	0.034
	0.1563	18	12	828.059	3	4.68	3	3.97	0.220
	0.1625	18	17	3.355	1	2.08	0	9.89	0.015
	0.1667	36	27	712.669	9	4.58	0	3.78	0.034
	0.1875	36	36	11.460	0	—	0	0.93	0.102
	0.2083	36	33	2.385	3	4.74	0	0.58	0.053

In terms of network complexity, the instances with $NC = 1.5$ were the ones with higher linear programming relaxation gaps and in which the solver found more difficulties in computing their optimal solutions. Regarding the skill factor, it is interesting to observe that despite the highest value of the linear programming relaxation gaps having been obtained for the instances with $SF = 0.5$, on average, that was not an obstacle for computing optimal solutions for these instances. In fact, the solver found the optimum for 53 instances (out of the 54 instances) associated with $SF = 0.5$ in approximately 11 seconds of CPU time. This behavior was also observed in instances with $MRS = 0.1625$ where a high linear programming relaxation gap did not prevent the solver from achieving the optimum of almost all the instances. The analysis of the remaining results from the perspective of the *MRS*, allows us to conclude that the worst performance in terms of both the linear programming relaxation gap and integer solving occurred in instances having $MRS = 0.1250$. From Table 3.6, we observe that these are the instances with fewer resources and this fact may justify the difficulty faced by the solver in attaining the optimum for these instances.

Table 5.19 contains the same information as Table 5.18 but for the discrete-time model P_{NDDT}.

From this table we observe that, not surprisingly, the instances became harder to solve optimally as the network complexity decreases. The instances with $NC = 1.5$ are the ones exhibiting the highest values in terms of the linear programming relaxation gaps and the CPU time required to solve them. As far as the skill factor is concerned, the results indicate again that, in spite of the instances with $SF = 0.5$

Table 5.19: Model P_{NDDT} — *Set 1*: results per each value of *SF*, *NC* and *MRS*.

		# inst	# opt	CPU (sec)	# tl	MIP gap (%)	# inf	*LP* gap (%)	*LP* CPU (sec)
SF	1	54	38	60.461	5	6.40	11	3.99	5.814
	0.75	54	37	1230.259	11	6.52	6	5.28	5.111
	0.5	54	40	654.305	8	7.79	6	8.72	3.103
	var.	54	35	685.319	8	7.47	11	6.06	6.347
NC	1.5	72	43	668.181	14	7.63	15	6.78	6.200
	1.8	72	48	369.277	13	4.99	11	5.59	5.122
	2.1	72	59	873.193	5	10.80	8	5.67	3.959
MRS	0.1250	72	25	2772.708	18	7.85	29	12.89	8.431
	0.1563	18	13	163.340	3	8.20	2	3.50	5.365
	0.1625	18	14	127.864	3	5.09	1	7.85	2.858
	0.1667	36	27	829.821	7	5.98	2	3.25	3.993
	0.1875	36	36	7.137	0	—	0	0.88	2.579
	0.2083	36	35	59.491	1	2.63	0	0.50	3.016

having the largest linear programming relaxation gaps, on average, the solver was able to achieve a larger percentage of optimal solutions for this set of instances than for the instances with other values of *SF*. The results further reveal that increasing the modified resource strength generally results in an increase of the percentage of instances solved optimally. The instances having $MRS = 0.1250$ are associated with the highest linear programming relaxation gaps and were some of the hardest to tackle by the solver (when looking for the optimal solution). Finally, a comparison between the results obtained by these two models allow us to conclude that the instances associated with $NC = 1.5$, $SF = 0.5$ or $MRS = 0.1250$ were the ones where the largest differences between the gaps of the linear programming relaxations produced by models P_{CT} and P_{NDDT} were observed. Therefore, it is for those sets of instances that the supremacy of the discrete-time models is larger.

Set 2

Next, we report a comparison of the results provided by the linear programming relaxations of all the models for instances in *Set 2*. For each instance, a time limit of 18000 seconds was considered for the resolution of the corresponding linear programming relaxation. We opted not to present any results regarding the performance of the models in terms of integer solving. The underlying reason for such decision is associated with the fact that the experiments performed on model P_{CT} with all valid inequalities and with the reduction test, i.e., the model that performed best for instances in *Set 1*, revealing the inability of the solver to provide a feasible solution for more than 88.8% of the instances in *Set 2* within 36000 seconds of CPU time. Some arguments that may justify these results include the randomness associated with CPLEX's heuristic (used at each node) and the introduction of additional valid inequalities both rendering a heavier model and not contributing to a significant improvement of the linear programming relaxation values. We believe that it is not worth considering any of the other models in terms of integer solving given the results obtained for *Set 1*.

Table 5.20 depicts the results in four columns. Column 1 refers to the models tested while Column 2 contains the number of instances, out of 216, for which the optimal solution of the linear programming relaxation was not computed within the stipulated time limit of 18000 seconds. Therefore, the rows of this table associated with models $P_M + (4.39 + 4.40)$ and P_{MDDT} contain average results for 213 instances, the rows associated with models P_{DDT} and $P_{MDDT} + (4.39 + 4.40)$ refer to the average results for 215 and

212 instances, respectively, while all the other rows refer to average results for 216 instances. The results are not surprising and follow the same pattern of the results already presented for instances in *Set 1*. Models P_{DDT}, P_{MDDT}, and P_{NDDT}, i.e., the discrete-time models with the disaggregate precedence constraints produced the best lower bounds but required a computational time incomparably larger than the time spent by the other models, in particular by the continuous-time models. Adding the valid inequalities (4.7) to model P_{CT} as well as the discretized versions of these constraints to the discrete-time formulations did not contribute to improving their respective linear programming relaxations and rendered, obviously, heavier models. When both the valid inequalities and the reduction test proposed by Correia et al. (2012) were added to model P_{CT}, a marginal improvement was achieved. Moreover, the computational time required to compute the linear programming relaxation of such model, was almost the double of its standard version (P_{CT}).

Table 5.20: Mathematical formulations — *Set 2*: overall results.

Model	#tl *LP*	*LP* gap (%)	*LP* CPU (sec)
P_{CT}	0	25.92	8.101
P_{CT} + (4.7)	0	25.92	15.491
P_{CT} + (4.7) + (*RT*) + (4.1 – 4.5)	0	25.51	15.103
P_{DT}	0	25.09	155.750
P_{DT} + (4.38)	0	25.09	396.269
P_{DDT}	1	21.92	607.345
P_{DDT} + (4.38)	0	21.96	1084.167
P_{M}	0	25.72	1643.995
P_{M} + (4.39 + 4.40)	3	25.62	2137.620
P_{MDDT}	3	21.78	1975.980
P_{MDDT} + (4.39 + 4.40)	4	21.70	2633.133
P_{N}	0	25.09	198.280
P_{N} + (4.39 + 4.40)	0	25.09	551.997
P_{NDDT}	0	21.96	764.064
P_{NDDT} + (4.39 + 4.40)	0	21.96	1299.823

Table 5.21: Model P_{CT} — *Set 2*: results per each value of *SF*, *NC* and *MRS*.

		# inst	*LP* gap (%)	*LP* CPU (sec)
SF	1	54	24.05	18.489
	0.75	54	25.09	6.684
	0.5	54	28.36	1.488
	var.	54	26.19	5.743
NC	1.5	72	32.12	7.891
	1.8	72	25.71	7.786
	2.1	72	19.93	8.625
MRS	0.0625	72	41.30	7.317
	0.0781	36	24.79	6.245
	0.0792	36	21.86	5.840
	0.0938	72	13.14	10.943

Similarly to what was done for *Set 1*, we focus our analysis on the models P_{CT} and P_{NDDT}. We recall that both models finished computing the linear programming relaxation for all the 216 instances

Table 5.22: Model P_{NDDT} — *Set 2*: results per each value of *SF*, *NC* and *MRS*.

		# inst	*LP* gap (%)	*LP* CPU (sec)
SF	1	54	20.57	1342.927
	0.75	54	21.38	645.859
	0.5	54	24.04	407.928
	var.	54	21.85	659.543
NC	1.5	72	27.13	1064.507
	1.8	72	21.85	722.594
	2.1	72	16.91	505.091
MRS	0.0625	72	34.07	1168.359
	0.0781	36	21.54	1037.814
	0.0792	36	18.89	586.426
	0.0938	72	11.60	311.713

of *Set 2*.

The analysis of Tables 5.21 and 5.22 allow us to conclude that in terms of skill factor (*SF*) both models originate the largest average linear programming relaxation gaps for instances with $SF = 0.5$ while the smallest ones are attained for instances with $SF = 1$. The instances having $SF = 1$ were also the most time consuming.

From the network complexity (*NC*) point of view, we observe that, for both models, the linear programming relaxation gap decreases as the network complexity (*NC*) increases. However, we observe that the computational time required to solve the linear programming relaxation of model P_{NDDT} decreases as the value of *NC* increases; for model P_{CT} we do not observe such behavior. In fact, the average time required by model P_{NDDT} for instances with $NC = 1.5$ is roughly the double of the time needed for the instances having $NC = 2.1$, whereas for model P_{CT}, the computational effort is roughly the same across the different *NC* values. It was also for the instances associated with $NC = 1.5$ that model P_{NDDT} produced the largest reduction in the average linear programming relaxation gaps, when compared with model P_{CT}, but at the expense of a large increase in the execution time.

Regarding the modified resource strength (*MRS*), one can observe that, for both models, the average gaps decrease as the *MRS* values increase. The average computational time required to solve the linear programming relaxation of model P_{NDDT} also decreases as the value of *MRS* increases. On the other hand, we observe that the instances associated with $MRS = 0.0938$ are, on average, the most time consuming for P_{CT}. The instances having $MRS = 0.0625$ were the ones where the use of model P_{NDDT} produced linear programming relaxation gaps that are roughly 17% smaller than the gaps produced by model P_{CT}. The instances in this class are among the ones that are the hardest to solve, due to a high resource scarcity.

Lower bounds

We now concentrate on the results provided by the two lower bound methods proposed in Section 4.3: *SbCB* and *SbCSLB*. For each method, we evaluate the quality of its associated lower bounds by computing a gap analogously to the linear programming relaxation gap defined previously. More specifically, the gap was computed as follows:

$$gap = \frac{Z^{UB} - \left\lceil Z^{lower} \right\rceil}{Z^{UB}} \times 100\%,$$

where Z^{UB} denotes the best available upper bound (considering the results provided by all the models

considered in this computational section and the approximate methods discussed previously) and Z^{lower} denotes the value computed by such method (either *SbCB* or *SbCSLB*).

We begin by discussing the results associated with the instances in *Set 1* and then we focus on the results achieved for the instances in *Set 2*.

Set 1

The results obtained by the two proposed lower bounds for the MSRCPSP are depicted in Table 5.23. This table consists of 5 columns. The instances are grouped by *SF*, *NC* and *MRS*. Column 2 indicates the specific value of the respective parameter depicted in the first column. Columns 3 and 4 illustrate the gap values provided by *SbCB* and *SbCSLB*, respectively. We recall, that over the course of the *SbCSLB*, it is necessary to verify whether some pairs of activities are compatible. The associated computational time required is depicted in Column 5. The total time required by *SbCB* and the additional time needed by *SbCSLB* were considered negligible and were omitted.

Table 5.23: Lower bounds — *Set 1*: *SbCB* and *SbCSLB* gaps.

		SbCB gap (%)	*SbCSLB* gap (%)	Pairs of compatible activities CPU (sec)
SF	1	53.65	4.03	0.078
	0.75	48.45	5.00	0.070
	0.5	45.94	7.04	0.041
	var.	49.01	5.28	0.074
NC	1.5	46.04	6.21	0.077
	1.8	48.62	4.99	0.068
	2.1	53.13	4.82	0.051
MRS	0.1250	43.22	11.16	0.051
	0.1563	53.21	3.97	0.075
	0.1625	45.36	5.32	0.036
	0.1667	49.14	3.63	0.073
	0.1875	55.04	0.85	0.074
	0.2083	55.67	0.58	0.088
Average		49.26	5.34	0.066

From Table 5.23, we observe that the results provided by both the *SbCB* and the *SbCSLB* follow an opposite tendency. In fact, as the values of the *SF*, *NC* or *MRS* increase, the average gaps provided by the *SbCB* tend to increase and the average gaps obtained by the *SbCSLB* generally decrease. The time spent checking weather two activities are compatible is almost the double for instances with $SF = 1$ than it is for instances with $SF = 0.5$. This observation is mostly associated with the fact that (i) the number of resources involved in the instances with $SF = 1$ is the double of the resources in instances having $SF = 0.5$ and (ii) the activities in instances with $SF = 1$ require every skill for their execution and hence every resource is eligible to contribute to any activity. This last aspect contributes to rendering heavier network flow problems when compared to the instances associated with smaller *SF* values.

From a *NC* point of view, the computational effort associated with checking if two activities are compatible increases as the value of this parameter decreases. For a fixed number of activities, the instances associated with smaller values of *NC* tend to allow a greater degree of parallelization (of the activities). Hence, there are potentially more pairs of overlapping activities whose compatibility has to be verified. This results in a higher number of flow problems to be solved, when compared to the instances with higher values of *NC*.

In terms of *MRS*, we observe a similar behavior to the results aggregated by *SF*. As the number of

resources involved in each instance increases (i.e., as the value of *MRS* becomes larger) the computational time required for computing pairs of compatible activities increases, in general.

The *SbCSLB* attains the best lower bound gaps of 5.34%, an average for the 216 instances, when compared to the *LP* gaps provided by each mathematical formulation presented in the Table 5.17 and to the lower bound gaps achieved by *SbCB*. Additional information reveals that such supremacy is observed for 31 instances (3 with *SF* = 1, 15 with *SF* = 0.5, 6 with *SF* = 0.75 and 7 with *SF* = var.) out of the 216 instances in this set. We would like to provide some additional insight regarding the *SbCB*. Such method was able to provide the best lower bound value for 6 out of the 216 instances that constitute *Set 1* (when compared to the values provided by the *LP* of all mathematical formulations and by the *SbCSLB*). We note that these 6 instances are associated with the smallest value of *MRS* in this set (*MRS* = 0.1250) and, as discussed previously, this type of instances is among the hardest to tackle. These 6 instances are partitioned per *SF* as follows: 4 with *SF* = 0.5, 1 with *SF* = var. and 1 with *SF* = 0.75.

We would like to emphasize that the behavior of the *SbCB* is strongly constrained by the number of available resources and the number of skills that they are proficient in. In fact, the *SbCB* relaxes the number of resources involved in the problem according to the skills they master (which ends up defining a problem much more simple than the original one). Moreover, the *SbCB* is dependent on the available resources, which are more abundant in the instances where activities require more skills for being processed, i.e., instances having higher values of *SF* and/or higher values of *MRS*. Since all instances in this set have 20 activities that demand on average 2 resources per each skill required for their execution, and consider the number of skills mastered by each resource to be pseudo-randomly generated between 1 and 3, a decrease on the number of resources (considering a fixed *SF* value) may have a direct influence in the denominator of *SbCB*—leading to a small cardinality of set $\mathcal{R}_l, l \in \mathcal{L}$ and hence to higher values (better lower bounds) provided by the *SbCB*.

Set 2

We now focus on the results obtained by the *SbCB* and the *SbCSLB* for the instances in *Set 2*. These results are presented in Table 5.24, which follows the same structure of Table 5.23.

Analyzing Table 5.24 from a *SF*, *NC* or *MRS* perspective yields the same conclusions that were drawn for *Set 1*. Hence, to avoid redundancy we focus our analysis on the results that are specific to *Set 2*.

Similarly to *Set 1*, the *SbCSLB* attains consistently better gaps than the *SbCB*. Unlike what was observed for *Set 1*, the best lower bounds were not provided by *SbCSLB*, which achieved a gap of 24.22%, on average. This fact justifies the effort to have developed new discrete-time formulations for the MSRCPSP. In fact, the linear programming relaxations of some of the referred formulations allowed to achieve better gaps (21.96%, on average) than the *SbCSLB*.

The time spent verifying the compatibility of some pairs of activities continues to be negligible (at most 1.3 CPU seconds, an average for the 54 instances with *SF* = 1) particularly when compared to the computational time associated with solving the linear programming relaxation of the mathematical models studied (see Table 5.20).

Additional information allows to conclude that the *SbCB* outperforms both the linear programming relaxations of all mathematical formulations and the lower bounds provided by the *SbCSLB* in 10 out of the 216 instances (3 instances with *SF* = 1, 4 instances with *SF* = 0.5, 1 instance with *SF* = 0.75 and 2 instances with *SF* = var.). The *SbCSLB* performed better than both the linear programming relaxation of all mathematical models and the *SbCB* in 20 out of the 216 instances (14 instances with *SF* = 0.5, 3 instances with *SF* = 0.75 and 3 instances with *SF* = var.).

We recall that the main purpose of this thesis was to develop effective approximate methods to compute feasible solutions to the MSRCPSP. When the heuristics proposed in thesis were developed, no

Table 5.24: Lower bounds — *Set 2*: *SbCB* and *SbCSLB* gaps.

		SbCB gap (%)	*SbCSLB* gap (%)	Pairs of compatible activities CPU (sec)
SF	1	46.54	23.84	1.335
	0.75	45.31	23.94	0.702
	0.5	42.86	24.48	0.392
	var.	45.75	24.62	0.688
NC	1.5	44.64	30.37	0.918
	1.8	44.25	23.89	0.800
	2.1	46.45	18.40	0.620
MRS	0.0625	44.22	37.71	0.509
	0.0781	43.88	22.35	0.912
	0.0792	46.35	21.46	0.705
	0.0938	46.01	13.05	1.020
Average		45.11	24.22	0.779

lower bound aside from the critical path length existed for the instances in *Set 2*. Later, and with the main purpose of improving those lower bounds, some effort was put into developing alternative mathematical formulations and lower bound methods for the problem. After performing the computational experiments just presented, it was possible to generally improve the lower bounds for both sets of instances. Hence, it becomes interesting to recompute the gaps for the instances in *Set 1* and to calculate the gaps for the instances in *Set 2*. For each instance and each approximate method, the gap is computed for each instance as follows:

$$gap = \frac{Z^{UB} - \left\lceil Z^{LB^*} \right\rceil}{Z^{LB^*}} \times 100\%,$$

where Z^{UB} denotes the best upper bound obtained by the heuristic method being evaluated and Z^{LB^*} is equal to the best known lower bound. We note that when assessing the quality of each BRKGA configuration studied, Z^{UB} will also take the value of the average upper bound achieved after performing the considered 5 runs.

For each instance, the value of Z^{LB^*} corresponds to the maximum of the lower bounds considering the best one obtained in Correia et al. (2012) and those provided by the studied mathematical formulations at CPLEX's termination, i.e., the optimal value or the best lower bound found before the stipulated time limit has been reached. We note that the values derived by *SbCSLB* and *SbCB* are implicitly considered in one of the continuous-time formulation studied, which receives information (as input) of both the best lower bound achieved by the two aforementioned methods and the linear programming relaxation values of all studied mathematical formulations. Hence, the lower bounds attained by such formulation at CPLEX's termination are never worse than the ones achieved by *SbCSLB* and *SbCB*.

Analogously to the computational results reported previously, we first present the (recomputed) gaps for the instances in *Set 1*, in Table 5.25, and then we present the gaps for the instances in *Set 2*, in Table 5.26. These tables follow the same structure. The first three columns identify the values of *SF*, *NC* and *MRS*. Column 4 refers to the gaps of the multi-pass heuristic, Columns 5–6 and Columns 7–8 depict the gaps for the $BRKGA_{PSS}$ and for the $BRKGA_{Both}$, respectively. More specifically, Columns 5 and 7 refer to the minimum gaps obtained and Columns 6 and 8 to the average gaps achieved. Each row corresponds to the average results for 6 instances (which are associated with the same values of *SF*, *NC* and *MRS*).

These results are then summarized in two tables that follow a similar structure to the one described above, Table 5.27 and Table 5.28, which contain the results for *Set 1* and *Set 2*, respectively.

Table 5.25: *Set 1* — Gaps of the developed heuristics computed using the best known lower bound.

			Multi-pass heuristic	BRKGA 5 runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil, p_e = 10\%, p_m = 30\%, \rho_e = 70\%, \mathcal{G} = \frac{n}{2}$			
				$\delta = PSS$		$\delta = Both$	
SF	*NC*	*MRS*	Gap (%)	Avg. gap (%)	Min. gap (%)	Avg. gap (%)	Min. gap (%)
1	1.5	0.1250	4.89	1.82	1.62	1.96	1.32
		0.1563	3.93	2.46	2.46	2.54	2.46
		0.1875	0.00	0.00	0.00	0.00	0.00
	1.8	0.1250	1.57	0.73	0.00	0.77	0.00
		0.1563	1.11	0.38	0.00	0.23	0.00
		0.1875	0.00	0.00	0.00	0.00	0.00
	2.1	0.1250	1.07	0.00	0.00	0.00	0.00
		0.1563	0.00	0.00	0.00	0.00	0.00
		0.1875	0.00	0.00	0.00	0.00	0.00
0.75	1.5	0.1250	3.98	0.70	0.53	0.85	0.26
		0.1667	0.90	0.33	0.00	0.44	0.00
		0.2083	1.79	0.70	0.44	1.05	0.88
	1.8	0.1250	6.47	2.04	0.82	2.27	1.31
		0.1667	1.15	0.61	0.00	0.71	0.31
		0.2083	0.00	0.00	0.00	0.00	0.00
	2.1	0.1250	3.99	0.95	0.00	0.74	0.22
		0.1667	0.00	0.00	0.00	0.24	0.00
		0.2083	0.79	0.00	0.00	0.00	0.00
0.5	1.5	0.1250	6.17	1.79	0.26	1.96	0.80
		0.1625	4.27	0.20	0.00	0.34	0.00
		0.1875	0.88	0.51	0.51	0.10	0.00
	1.8	0.1250	5.19	0.79	0.34	0.97	0.63
		0.1625	2.42	0.29	0.00	0.41	0.00
		0.1875	0.36	0.14	0.00	0.22	0.00
	2.1	0.1250	5.00	0.42	0.00	0.39	0.24
		0.1625	0.93	0.00	0.00	0.00	0.00
		0.1875	0.00	0.00	0.00	0.00	0.00
var.	1.5	0.1250	7.55	3.12	1.58	3.45	2.19
		0.1667	1.84	1.27	1.11	1.40	1.11
		0.2083	0.91	0.00	0.00	0.00	0.00
	1.8	0.1250	5.16	0.91	0.00	1.47	0.93
		0.1667	2.31	0.38	0.38	0.51	0.00
		0.2083	0.00	0.71	0.51	0.99	0.00
	2.1	0.1250	2.46	0.38	0.32	0.32	0.32
		0.1667	0.33	0.52	0.00	0.67	0.41
		0.2083	0.00	0.00	0.00	0.00	0.00
Average			2.15	0.62	0.30	0.70	0.37

Table 5.26: *Set 2* — Gaps of the developed heuristics computed using the best known lower bound.

			Multi-pass heuristic	BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil, p_e = 10\%, p_m = 30\%, \rho_e = 70\%, \mathcal{G} = \frac{n}{2}$			
				$\delta = PSS$		$\delta = Both$	
SF	*NC*	*MRS*	Gap (%)	Avg. gap (%)	Min. gap (%)	Avg. gap (%)	Min. gap (%)
1	1.5	0.0625	78.17	69.21	67.65	69.60	67.85
		0.0781	43.58	39.20	37.21	39.25	37.52
		0.0938	14.62	12.54	11.59	12.15	10.54
	1.8	0.0625	65.89	56.65	55.06	57.01	56.02
		0.0781	25.93	20.78	19.22	20.26	19.17
		0.0938	13.39	12.06	11.01	11.71	10.73
	2.1	0.0625	47.63	40.96	39.77	40.75	39.52
		0.0781	17.79	14.75	14.19	15.23	14.21
		0.0938	9.78	8.61	7.97	8.45	7.70
0.75	1.5	0.0625	79.61	69.83	68.57	68.91	67.83
		0.0792	45.72	40.42	38.67	40.44	37.99
		0.0938	24.81	21.35	20.25	21.30	19.76
	1.8	0.0625	68.61	55.39	53.92	54.26	51.76
		0.0792	30.71	26.01	24.39	25.88	23.86
		0.0938	14.97	12.72	10.88	12.52	11.18
	2.1	0.0625	45.01	35.54	33.89	35.13	33.19
		0.0792	17.33	14.45	13.36	14.49	12.68
		0.0938	10.56	10.15	9.45	9.99	9.17
0.5	1.5	0.0625	68.03	48.47	44.77	44.37	41.40
		0.0781	48.44	37.10	34.42	35.31	32.87
		0.0938	31.36	27.50	25.91	27.70	26.16
	1.8	0.0625	53.74	36.29	33.98	36.31	34.46
		0.0781	35.62	26.21	24.02	26.80	24.52
		0.0938	26.53	19.41	17.77	19.43	17.40
	2.1	0.0625	44.28	31.18	29.08	29.73	26.74
		0.0781	24.85	17.81	16.72	17.45	15.08
		0.0938	8.19	5.94	5.64	5.82	5.38
var.	1.5	0.0625	79.13	68.00	65.50	64.94	62.71
		0.0792	36.09	30.13	27.65	30.14	28.13
		0.0938	25.07	22.68	21.32	22.79	21.22
	1.8	0.0625	55.09	42.79	40.09	40.83	38.43
		0.0792	28.12	25.29	23.70	25.14	23.27
		0.0938	24.12	20.36	18.76	20.71	19.41
	2.1	0.0625	39.92	32.92	30.94	32.08	30.10
		0.0792	23.61	20.96	19.82	20.89	19.79
		0.0938	10.57	9.36	7.94	9.71	8.70
Average			36.58	30.08	28.47	29.65	27.96

Table 5.27: *Set 1* — Gaps of the developed heuristics computed using the best known lower bound, summary.

		Multi-pass heuristic	BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil, p_e = 10\%, p_m = 30\%, \rho_e = 70\%, \mathcal{G} = \frac{n}{2}$			
			$\delta = PSS$		$\delta = Both$	
		Gap (%)	Avg. gap (%)	Min. gap (%)	Avg. gap (%)	Min. gap (%)
SF	1	1.40	0.60	0.45	0.61	0.42
	0.75	2.12	0.59	0.20	0.70	0.33
	0.5	2.80	0.46	0.12	0.49	0.19
	var.	2.28	0.81	0.43	0.98	0.55
NC	1.5	3.09	1.08	0.71	1.18	0.75
	1.8	2.15	0.58	0.17	0.71	0.27
	2.1	1.21	0.19	0.03	0.20	0.10
MRS	0.1250	4.46	1.14	0.46	1.26	0.68
	0.1563	1.68	0.95	0.82	0.92	0.82
	0.1625	2.54	0.16	0.00	0.25	0.00
	0.1667	1.09	0.52	0.25	0.66	0.30
	0.1875	0.21	0.11	0.08	0.05	0.00
	0.2083	0.58	0.23	0.16	0.34	0.15
Average		2.15	0.62	0.30	0.70	0.37

Table 5.28: *Set 2* — Gaps of the developed heuristics computed using the best known lower bound, summary.

		Multi-pass heuristic	BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil, p_e = 10\%, p_m = 30\%, \rho_e = 70\%, \mathcal{G} = \frac{n}{2}$			
			$\delta = PSS$		$\delta = Both$	
		Gap (%)	Avg. gap (%)	Min. gap (%)	Avg. gap (%)	Min. gap (%)
SF	1	35.20	30.53	29.30	30.49	29.25
	0.75	37.48	31.76	30.37	31.44	29.71
	0.5	37.89	27.77	25.81	26.99	24.89
	var.	35.75	30.28	28.41	29.69	27.97
NC	1.5	47.88	40.53	38.62	39.74	37.83
	1.8	36.89	29.50	27.73	29.24	27.52
	2.1	24.96	20.22	19.06	19.98	18.52
MRS	0.0625	60.43	48.94	46.93	47.83	45.83
	0.0781	32.70	25.98	24.30	25.72	23.89
	0.0792	30.26	26.21	24.60	26.16	24.29
	0.0938	17.83	15.22	14.04	15.19	13.94
Average		36.58	30.08	28.47	29.65	27.96

We observe that the results presented in the previous sections (with the exception of PR_m, which is not influenced by lower bound improvements) were highly overestimated, particularly for *Set 1*. Nonetheless, the conclusions drawn therein seem to hold. By improving the lower bounds for some of the instances that remained unsolved by Correia et al. (2012), the average gaps obtained by the $BRKGA_{PSS}$ for 216 instances of *Set 1* were reduced to 0.30%, reinforcing the quality of the proposed heuristics.

5.4 Conclusions

This chapter presents the main contributions of this thesis, which concern the development of approximate methods for computing feasible solutions to the problem being studied.

With the purpose of avoiding random resource selection and allocation (both to skill and activity), we introduced the concepts of activity grouping and resource weight. Instead of simply determining whether a feasible flow exists in a specific network—a fundamental condition for a set of activities to be processed simultaneously at a specific moment in time, we suggested going one step further by finding a set of resources that meets their skill demands with the least total cost. We believe that by aggregating the requirements of a set of activities eligible to be scheduled at a given time and fulfilling them at once through the assignment of the referred set of resources, a more proper resource allocation may be achieved.

The two well-known parallel and serial schedule generation schemes (PSS and SSS, respectively) were extended to our problem setting and the two concepts referred above were incorporated into them. We revisited the notion of scheduling the project from the end to the beginning, which is equivalent to reversing all the arcs in the precedence network. The main purpose for considering the reversed precedence network in the heuristics developed is associated with the possibility of such representation of the precedence network allowing to achieve makespan values different from those obtained when the original precedence network is used. Two new heuristics were proposed, namely a multi-pass heuristic built upon the developed parallel scheduling scheme and a biased random-key genetic algorithm. Regarding the multi-pass heuristic, we considered several single and multi-priority rules for selecting the activities to be scheduled at each moment in time and we developed 3 resource weight functions for computing the weights of the resources. After introducing a general framework for the biased-random key genetic algorithm, we defined the problem-dependent components, namely the chromosome structure and the decoders. By incorporating a wide range of characteristics, including those specific to our problem, into the chromosomes (e.g., resource weights, type of decoder and precedence network schemes to be used), a comprehensive chromosome structure was derived.

Extensive computational experiments were performed. The heuristics proposed in this chapter and the methodologies presented in Chapter 4 were tested on a total of 432 instances of the problem, partitioned into two sets: *Set 1* and *Set 2* (see Sections 3.4.2 and 3.4.4, respectively, for detailed descriptions of the instances considered). The results allowed us to conclude that the complexity of this problem justifies the indispensable use of procedures based on heuristics to compute good quality feasible solutions to real problems in an admissible time. In fact, a heuristic that provides better upper bounds, even at the cost of increased computational time, is often preferred when compared to a faster heuristic that provides worse upper bounds, since real-world instances of these problems are expected to be tackled only by approximate methods.

The multi-pass heuristic proved to be an efficient approach time-wise. In fact, it provided good quality solutions within a very small computational time. For the larger instances, we observed that the computational time required by an off-the-shelf solver for achieving feasible solutions of the same quality of those provided by the multi-pass heuristic was prohibitive. In fact, the solver failed to attain a feasible solution for more than 72% of the instances in *Set 2*.

The computational results of the BRKGA indicated that the configuration with the single decoder

PSS provides the best results for the smaller-sized instances whereas the BRKGA with the two decoders, the PSS and SSS, achieves the best results for the set of larger instances. Overall, we conclude that the upper bounds computed by the multi-pass heuristic were greatly improved by both BRKGA configurations tested. Furthermore, we observed that the proposed BRKGA is extremely robust, in the sense that it has the ability to find solutions of a similar quality (for a given instance) in distinct runs.

The numerical experiments performed on the mathematical formulations studied showed that the use of a solver is more suitable, as expected, for instances of small dimensions. In order to evaluate the quality of the lower bounds produced by the different models (stemming from their respective linear programming relaxations) we performed a series of computational experiments on the two sets of benchmark instances from the literature used in the previous experiments. Additionally, we deepened the analysis by focusing it in the comparison between the continuous-time model and one of the studied discrete-time models (with disaggregated precedence constraints). Such comparison allowed us to identify the features of the test instances that highlight the supremacy of the referred discrete-time model in terms of the quality of the lower bounds stemming from its linear programming relaxation. Finally, we also analyzed the suitability of each model to solve the small-sized instances to optimality, using a general purpose solver. Although the discrete-time models provided the best linear programming relaxation gaps, the continuous-time models achieved the most optimal solutions. Such conclusion had already been drawn by Koné et al. (2011) for the RCPSP.

In terms of the developed lower bound mechanisms we observe that the *SbCSLB* provided good results by attaining the best lower bound gaps, on average, for instances in *Set 1*. Regarding *Set 2*, the best lower bounds were achieved, on average, by some discrete-time formulations.

After performing the computational experiments on both the mathematical formulations studied and the two derived lower bound methods, it was possible to improve the quality of the existing lower bounds for some instances of *Set 1* and to obtain lower bounds of a better quality than the critical path length for the instances in *Set 2*. This led us to calculate a meaningful gap between the upper bounds obtained by the proposed approximate methods for the instances in *Set 2* and to recompute the gaps for the instances in *Set 1*.

CHAPTER 6

Conclusions and directions for future work

In this thesis, we investigated a particular project scheduling problem with multi-skill resources. This problem usually arises in companies that deal with complex projects whose requirements have to be fulfilled by human resources organized in multidisciplinary teams.

After reviewing the work published so far in this field of research, it became clear that some effort could still be made in terms of developing competitive approximate methods for the problems considering homogeneous resources.

The specific problem studied, which considers the resources as being homogeneous, was fully described. It is important to note that aside from a set of instances (of small dimensions), there were no properly generated or described instances in the literature for the specific problem studied. This fact motivated the development of the instance generator formally proposed in this thesis. Using such generator, a new set of larger instances was built. The characteristics of these two sets of instances were thoroughly described.

Several mathematical models, lower bound mechanisms and heuristic procedures were proposed. The main purpose of developing alternative mathematical formulations for the problem was associated with the need to improve the existing lower bounds, which consisted of the critical path length for the larger instances. Two new optimization models and two new lower bounds were derived. The computational experiments revealed, as expected, the unsuitability of the mathematical models to cope with the increased complexity of, particularly, large-sized instances. These results corroborated our decision to focus on the development of approximate methods for the problem.

The development of approximate methods for this problem is of major relevance if one thinks that the real-world project scheduling problems faced by practitioners are of large dimensions and that the use of exact methods often becomes impracticable for problems of such size. The development of effective approximate solution methodologies consists in one of the major contributions of this thesis. The well-known parallel and serial scheduling schemes, which are the backbone for development of heuristics for project scheduling problems, were extended to our problem setting. Then, we thoroughly described the two heuristics proposed: a multi-pass heuristic built upon such parallel scheduling scheme and a biased random-key genetic algorithm with different configurations.

The computational results revealed that the proposed multi-pass heuristic is more time-efficient and the biased random-key genetic algorithm achieves solutions of higher quality, on average. In terms of the mathematical formulations, it is observed that the continuous-time formulations studied

are more suitable for solving small- and medium-sized instances of the problem optimally, while the analyzed discrete-time formulations yielded stronger linear programming relaxations, as expected. The two proposed lower bounds methods improved the lower bounds for some instances. Despite the use of heuristics being more suitable for solving this type of problems, the development of mathematical formulations continues also to be of great importance since there are some instances, namely those of relatively small size, whose characteristics make them easy to tackle by an off-the-shelf solver.

The problem investigated in this thesis has been scarcely treated in the literature. Therefore, the contributions herein represent a relevant step forward towards the development of new and more efficient procedures for tackling instances of a realistic dimension. Nonetheless, there is still a wide range of methodologies and extensions of the problem investigated that remain unstudied.

We begin by introducing some suggestions for further research regarding methodological developments and then we discuss some potentially interesting extensions of the addressed problem that could be investigated.

A research direction that is highlighted in this work regards the development of effective tools for computing sharp lower bounds which are especially important for large-scale instances of the problem. Lower bounds allow a more accurate evaluation of the quality of feasible solutions provided by heuristics and may contribute to improve the performance of exact methods. Hence, it is of major relevance to continue the research for fast algorithms for deriving good lower bounds for this problem. The development of alternative mathematical formulations should also be pursued. In fact, depending on their structures, new optimization models may be used to compute feasible solutions and even to solve some medium- and large-sized instances of the problem as well as to yield lower bounds stemming from their linear programming relaxations.

In terms of approximate methods, we believe that there is still an opportunity to derive both new heuristics and local search mechanisms. By exploiting some crucial features of the problem, the heuristics proposed in this thesis took the research in this area one step further and proved to be competitive for solving instances of the problem of a reasonable size. Nonetheless, the mechanisms used in the developed heuristics may have also opened new directions for further research. More specifically, and in the context of priority-based heuristics (e.g., the developed multi-pass heuristic) future work may concern the development of new deterministic functions to compute the weights of the resources, with a particular emphasis to those whose weights computed to each resource may vary over time. These dynamic weight rules may consider instance-specific information related to, for instance, the skill requirements and processing times of activities not yet scheduled. Regarding the developed biased random-key genetic algorithm, it would be interesting to study other crossover operators, solution encodings as well as different and possibly more comprehensive chromosome structures. Moreover, the investigation of other stopping criteria is also of particular interest. In fact, other stopping criteria may cause the time spent by the algorithm to decrease while possibly not compromising the quality of the solutions achieved. A common denominator for both approximate methods is the procedure for assigning the resources to the activities. Currently, the resources are assigned to the activities in a two-step process: first, we determine the skill that each resource will perform by finding a min-cost flow in a specific network; then, the resources with higher weights are assigned to the activities with smaller processing times as much as possible. It would be interesting to evaluate the computational effort required to solve this problem at once by considering, for instance, the second set of nodes in the graph depicted in Figure 5.1 to be associated with the specific pairs (activity, skill) instead of simply considering the skills required to process the activities. Theoretically, this will render a much more complex graph structure and consequently, a heavier min-cost flow problem to be solved.

Regarding the type of problem, we observe that several variants of project scheduling problems with

multi-skill resources remain unstudied.

By looking into Tables 2.1 and 2.2, we realize that little work has been done on multi-objective problems. More specifically, we point out that the two most well-studied objective functions in project scheduling problems which are often conflicting: the makespan of the project and the cost for its execution have not been yet considered together in the context of project scheduling problems with multi-skill resources. Due to its particular interest for real-world problems, the study of that specific multi-objective function in the scope of project scheduling problems with multi-skill resources (either homogeneous or non-homogeneous) constitutes a possible direction for further research.

After examining the aforementioned tables it also becomes clear that the references considering homogeneous resources always study a single project and a single objective function. Hence, it may be interesting to consider multi-project environments and multi-objective functions. In terms of problems considering non-homogeneous resources, we suggest the extension of some of the concepts introduced in this thesis (e.g., resource weight, the notion of meeting the requirements of a set of activities at once) to account for the heterogeneity of the resources, as an attempt to better capture the specific features of those problems.

Activity preemption is a feature that has been widely studied in the context of resource-constrained project scheduling problems and consists of another proposal for a deeper investigation when it comes to considering problems with multi-skill resources. This extension illustrates real-world situations where activities have occasionally to be halted. We note that the preemptive version of a problem involving non-homogeneous resources has not been addressed to date and hence constitutes a possible direction for further research.

The majority of the analyzed research consider that the processing times of the activities are fixed. It may make sense to consider variable processing times for the activities, namely in the presence of non-homogeneous resources. In that situation, we may conclude that the smallest processing time of an activity is achieved when the set of resources assigned to each skill it requires consists of the most proficient ones in each of those skills.

The problems analyzed by the reviewed research are deterministic since all the information concerning the problem is available and known beforehand and do not change over time. However, in real-world settings, this is almost never the case. Hence, a plausible research direction may include the investigation of project scheduling problems with multi-skill resources under uncertainty. This concept has already been studied in the context of resource-constrained project scheduling problems by considering stochastic processing times for the activities, i.e., the processing times of the activities are random numbers which follow a specific probability distribution function.

Bibliography

Alba, E. and Chicano, J. F. (2007). Software project management with GAs. *Information Sciences*, 177:2380–2401.

Alcaraz, J. and Maroto, C. (2001). A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, 102(1):83–109.

Almeida, B. F., Correia, I., and Saldanha-da-Gama, F. (2015). An Instance Generator for the Multi-Skill Resource-Constrained Project Scheduling Problem. Technical report, Faculdade de Ciências da Universidade de Lisboa — Centro de Matemática, Aplicações Fundamentais e Investigação Operacional. Available at `https://ciencias.ulisboa.pt/sites/default/files/fcul/unidinvestig/cmaf-cio/SGama.pdf`.

Almeida, B. F., Correia, I., and Saldanha-da-Gama, F. (2016a). A BRKGA for the project scheduling problem with flexible resources — Submitted. Working Paper available at `http://cmafcio.campus.ciencias.ulisboa.pt/sites/cmafcio/files/A%20BRKGA%20for%20the%20project%20scheduling%20problem%20with%20flexible%20resources.pdf`.

Almeida, B. F., Correia, I., and Saldanha-da-Gama, F. (2016b). Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, 57:91–103.

Alvarez-Valdés, R. and Tamarit, J. M. (1993). The project scheduling polyhedron: Dimension, facets and lifting theorems. *European Journal of Operational Research*, 67(2):204–220.

Artigues, C. (2017). On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154–159.

Artigues, C., Koné, O., Lopez, P., and Mongeau, M. (2015). Mixed-integer linear programming formulations. In Schwindt, C. and Zimmermann, J., editors, *Handbook on Project Management and Scheduling*, volume 1, pages 17–41. Springer International Publishing.

Baptiste, P., Pape, C. L., and Nuijten, W. (1999). Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92:305–333.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160.

Bellenguez-Morineau, O. and Néron, E. (2005). Lower Bounds for the Multi-skill Project Scheduling Problem with Hierarchical Levels of Skills. In *Proceedings of the 5th international conference on Practice and Theory of Automated Timetabling*, PATAT'04, pages 229–243.

Bellenguez-Morineau, O. and Néron, E. (2007). A Branch-and-Bound Method for Solving Multi-Skill Project Scheduling Problem. *RAIRO Operations Research*, 41:155–170.

Błażewicz, J., Lenstra, J. K., and Kan, A. H. G. R. (1983). Scheduling Subject to Resource Constraints - Classification and Complexity. *Discrete Applied Mathematics*, 5:11–24.

Brooks, G. and White, C. (1965). An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *Journal of Industrial Engineering*, 16:34–40.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41.

Carlier, J. and Néron, E. (2003). On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149(2):314–324.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35:164–176.

Christofides, N., Alvarez-Valdes, R., and Tamarit, J. (1987). Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29:262–273.

Correia, I., Lourenço, L. L., and Saldanha-da-Gama, F. (2012). Project scheduling with flexible resources: formulation and inequalities. *OR Spectrum*, 34:635–663.

Correia, I. and Saldanha-da-Gama, F. (2014). The impact of fixed and variable costs in a multi-skill project scheduling problem: An empirical study. *Computers & Industrial Engineering*, 72:230–238.

Correia, I. and Saldanha-da-Gama, F. (2015a). A modeling framework for project staffing and scheduling problems. In Schwindt, C. and Zimmermann, J., editors, *Handbook on Project Management and Scheduling*, volume 1, pages 547–564. Springer International Publishing.

Correia, I. and Saldanha-da-Gama, F. (2015b). A note on "branch-and-price approach for the multi-skill project scheduling problem". *Optimization Letters*, 9(6):1255–1258.

Demassey, S. (2008). Mathematical programming formulations and lower bounds. In Artigues, C., Demassey, S., and Néron, E., editors, *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, pages 49–61. ISTE Ltd and John Wiley & Sons.

Demeulemeester, E. L. and Herroelen, W. (2002). *Project Scheduling: A Research Handbook*. Springer.

Demeulemeester, E. L. and Herroelen, W. S. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11):1485–1492.

Dhib, C., Soukhal, A., and Néron, E. (2015). Mixed-integer linear programming formulation and priority-rule methods for a preemptive project staffing and scheduling problem. In Schwindt, C. and Zimmermann, J., editors, *Handbook on Project Management and Scheduling*, volume 1, pages 603–617. Springer International Publishing.

Drezet, L.-E. and Billaut, J.-C. (2008). A project scheduling problem with labour constraints and time-dependent activities requirements. *International Journal of Production Economics*, 112:217–225.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc.

Gonçalves, J. F., Mendes, J. J., and Resende, M. G. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171–1190.

Gonçalves, J. F. and Resende, M. G. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487–525.

Gonçalves, J. F. and Resende, M. G. (2013). A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics*, 145(2):500–510.

Gonçalves, J. F. and Resende, M. G. (2015). A biased random-key genetic algorithm for the unequal area facility layout problem. *European Journal of Operational Research*, 246(1):86–107.

Gonçalves, J. F., Resende, M. G., and Mendes, J. J. (2011). A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics*, 17(5):467–486.

Gutjahr, W. J., Katzensteiner, S., Reiter, P., Stummer, C., and Denk, M. (2008). Competence-driven project portfolio selection, scheduling and staff assignment. *Central European Journal of Operations Research*, 16:281–306.

Gutjahr, W. J., Katzensteiner, S., Reiter, P., Stummer, C., and Denk, M. (2010). Multi-objective decision analysis for competence-oriented project portfolio selection. *European Journal of Operational Research*, 205:670–679.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, 49(5):433–448.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14.

Heimerl, C. and Kolisch, R. (2010). Scheduling and staffing multiple projects with a multi-skilled workforce. *OR Spectrum*, 32:343–368.

Herroelen, W., De Reyck, B., and Demeulemeester, E. (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25:279–302.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Kelley Jr., J. E. and Walker, M. R. (1959). Critical-path planning and scheduling. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, IRE-AIEE-ACM '59 (Eastern), pages 160–173.

Klein, R. (2000). Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects. *European Journal of Operational Research*, 127:619–638.

Klein, R. and Scholl, A. (1999). Computing lower bounds by destructive improvement: an application to resource-constrained project scheduling problem. *European Journal of Operational Research*, 112:322–346.

Kolisch, R. (1995). *Project Scheduling Under Resource Constraints: Efficient Heuristics for Several Problem Classes*. Physica-Verlag Heidelberg.

Kolisch, R. (1996a). Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14:179–192.

Kolisch, R. (1996b). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333.

Kolisch, R. and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174:23–37.

Kolisch, R. and Heimerl, C. (2012). An Efficient Metaheuristic for Integrated Scheduling and Staffing IT Projects Based on a Generalized Minimum Cost Flow Network. *Naval Research Logistics*, 59:111–127.

Kolisch, R. and Sprecher, A. (1996). PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216.

Koné, O., Artigues, C., Lopez, P., and Mongeau, M. (2011). Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38:3–13.

Li, H. and Womer, K. (2009). Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling*, 12:281–298.

Li, K. and Willis, R. (1992). An alternative scheduling technique for resource constrained project scheduling. *European Journal of Operational Research*, 56:370–379.

Malcolm, D. G., Roseboom, J. H., Clark, C. E., and Fazar, W. (1959). Application of a technique for research and development program evaluation. *Operations Research*, 7(5):646–669.

Mendes, J. J., Gonçalves, J. F., and Resende, M. G. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research*, 36(1):92–109.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998). An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44(5):714–729.

Montoya, C., Bellenguez-Morineau, O., Pinson, E., and Rivreau, D. (2014). Branch-and-price approach for the multi-skill project scheduling problem. *Optimization Letters*, 8(5):1721–1734.

Néron, E. (2002). Lower Bounds for the Multi-Skill Project Scheduling Problem. In *Proceeding of the Eighth International Workshop on Project Management and Scheduling*, pages 274–277.

Özdamar, L. (1999). A genetic algorithm approach to a general category project scheduling problem. *IEEE Transactions on Systems, Man and Cybernetics*, 29:44–69.

Pritsker, A. A. B., Watters, L. J., and Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16:93–108.

Ruiz, E., Albareda-Sambola, M., Fernández, E., and Resende, M. G. (2015). A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. *Computers & Operations Research*, 57(C):95–108.

Schwindt, C. and Zimmermann, J. (2015). *Handbook on Project Management and Scheduling Vol.1*. Springer.

Smith-Miles, K. and Bowly, S. (2015). Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113.

Spears, W. M. and Jong, K. A. D. (1991). On the virtues of Parameterized Uniform Crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236.

Stinson, J. P., Davis, E. W., and Khumawala, B. M. (1978). Multiple resource-constrained scheduling using branch and bound. *A I I E Transactions*, 10(3):252–259.

Węglarz, J., Józefowska, J., Mika, M., and Waligóra, G. (2011). Project scheduling with finite or infinite number of activity processing modes - a survey. *European Journal of Operational Research*, 208:177–205.

Yannibelli, V. and Amandi, A. (2011). A knowledge-based evolutionary assistant to software development project scheduling. *Expert Systems with Applications*, 38(7):8403–8413.

Yannibelli, V. and Amandi, A. (2013). Hybridizing a multi-objective simulated annealing algorithm with a multi-objective evolutionary algorithm to solve a multi-objective project scheduling problem. *Expert Systems with Applications*, 40(7):2421–2434.

CHAPTER 7

Appendix

This appendix contains the pseudo-codes for the instance generator presented in Section 3.4.3. In particular, we formalize 5 algorithms properly introduced in the referred section. We note that whenever we refer to a parameter as being randomly generated in a set $\{a, \ldots, b\}$, this means that the parameter was generated according to a discrete uniform distribution in that set.

Algorithm 7.1: Generating a precedence network — Step 1

Data: $nStart$, $nFinish$, $MaxSucc$, $Pred(j) : j \in V$, $Succ(j) : j \in V$
Result: Randomly selects a predecessor i for each non-dummy activity $j \in V : Pred(j) = \emptyset$

$nonredarcs \longleftarrow nStart + nFinish$; // number of non-redundant arcs in the graph;
$j \longleftarrow nStart + 1$;
while $j < n + 1$ **do**
    while $Pred(j) = \emptyset$ **do**
        if $j \geq (n - nFinish + 1)$ **then**
            $i \longleftarrow random \in \{1, ..., (n - nFinish)\}$;
        else
            $i \longleftarrow random \in \{1, ..., j - 1\}$;
        end
        if $\left|Succ(i)\right| < MaxSucc$ **then**
            $Succ(i) \longleftarrow Succ(i) \cup \{j\}$;
            $Pred(j) \longleftarrow \{i\}$;
            $nonredarcs \longleftarrow nonredarcs + 1$;
        end
    end
    $j \longleftarrow j + 1$;
end

Algorithm 7.2: Generating a precedence network — Step 2

Data: $nStart, nFinish, MaxPred, nonredarcs, Pred(j) : j \in V, Succ(j) : j \in V$
Result: Randomly selects a successor u for each non-dummy activity $j \in V : Succ(j) = \emptyset$

$j \longleftarrow n - nFinish$;
while $j > 0$ **do**
  Compute $\overline{Pred}(j)$;
  while $Succ(j) = \emptyset$ **do**
    if $j \leq nStart$ **then**
      $u \longleftarrow random \in \{nStart + 1, ..., n\}$;
    else
      $u \longleftarrow random \in \{j + 1, ..., n\}$;
    end
    if $\left|Pred(u)\right| < MaxPred \wedge u \notin \cup_{i \in \overline{Pred}(j)} Succ(i)$ **then**
      $Pred(u) \longleftarrow Pred(u) \cup \{j\}$;
      $Succ(j) \longleftarrow \{u\}$;
      $nonredarcs \longleftarrow nonredarcs + 1$;
    end
  end
  $j \longleftarrow j - 1$;
end

Algorithm 7.3: Generating a precedence network — Step 3

Data: $nStart, nFinish, Pred(j) : j \in V, Succ(j) : j \in V, MaxPred, MaxSucc, nonredarcs, NC$
Result: Creates a network with the desired NC

$reqnumarcs := \lceil NC \times (n + 1) \rceil$ // required number of arcs to fulfill the desired NC;
if $nonredarcs > reqnumarcs$ **then**
  Go to Algorithm 7.4;
else
  while $nonredarcs \leq reqnumarcs$ **do**
    $i \longleftarrow random \in \{1, \ldots, (n - nFinish)\}$;
    if $\left|Succ(i)\right| < MaxSucc$ **then**
      if $i \leq nStart$ **then**
        $j \longleftarrow random \in \{nStart + 1, \ldots, n\} \setminus Succ(i)$;
      else
        $j \longleftarrow random \in \{i + 1, \ldots, n\} \setminus Succ(i)$;
      end
    end
    if $\left|Pred(j)\right| < MaxPred$ **then**
      if $j \in \overline{Succ(i)} \vee \exists\, u \in \overline{Succ(j)} : Pred(u) \cap \overline{Pred(i)} \neq \emptyset$
        $\vee Pred(j) \cap \overline{Pred(i)} \neq \emptyset \vee Succ(i) \cap \overline{Succ(j)} \neq \emptyset$ **then**
        This arc is not added as it creates redundancy in the network;
      else
        $Pred(j) \longleftarrow Pred(j) \cup \{i\}$;
        $Succ(i) \longleftarrow Succ(i) \cup \{j\}$;
        $nonredarcs \longleftarrow nonredarcs + 1$;
      end
    end
  end
end

Algorithm 7.4: Generating a precedence network — Step 3: removing arcs from the network

Data: $nFinish$, $Pred(j) : j \in V$, $Succ(j) : j \in V$, $nonredarcs$, $reqnumarcs$
Result: Removes arcs from the network until the desired NC is reached

while $nonredarcs > reqnumarcs$ **do**
  $i \longleftarrow random \in \{1, ..., (n - nFinish)\}$;
  if $\left|Succ(i)\right| > 1$ **then**
    $j \longleftarrow random \in Succ(i)$;
    if $\left|Pred(j)\right| > 1$ **then**
      $Succ(i) \longleftarrow Succ(i) \setminus \{j\}$;
      $Pred(j) \longleftarrow Pred(j) \setminus \{i\}$;
      $nonredarcs \longleftarrow nonredarcs - 1$;
    end
  end
end

Algorithm 7.5: Generation of the activities — Step 2

Data: V; $\mathcal{L}$; $\mathcal{L}_j, j \in V$; SF
Result: Generation of the skill requirements

$\mathcal{L}_0, \mathcal{L}_{n+1} \longleftarrow \emptyset$;
if $SF \in]0, 1]$ **then**
  $\lambda \longleftarrow \lceil SF \times L \rceil$ // λ:= number of skills required by each activity $j \in V \setminus \{0, n+1\}$;
end
while *There is at least one skill not required by any activity* **do**
  $j \longleftarrow 1$;
  $\rho \longleftarrow 0$ // ρ := overall number of already associated resources;
  while $j < n + 1$ **do**
    $\mathcal{L}_j \longleftarrow \emptyset$;
    if $SF \notin]0, 1]$ **then**
      $\lambda \longleftarrow random \in \{2, \ldots, L\}$; // a non-valid SF value has been provided, a variable SF is considered;
    end
    while $\left|\mathcal{L}_j\right| < \lambda$ **do**
      $l \longleftarrow random \in \mathcal{L}$;
      if $l \notin \mathcal{L}_j$ **then**
        $\mathcal{L}_j \longleftarrow \mathcal{L}_j \cup \{l\}$;
        $r_{jl} \longleftarrow 1$;
        $\rho \longleftarrow \rho + 1$;
      end
    end
    $j \longleftarrow j + 1$;
  end
end

Algorithm 7.6: Generation of the activities — Step 3

Data: V, $\mathcal{R}$, $\mathcal{L}_j, j \in V$, MRS, ρ, $maxResSkill$, $maxResAct$
Result: Increase the requirements of the activities until MRS is reached

while $\rho < \left\lfloor \frac{K}{MRS} \right\rfloor$ **do**
  $j \longleftarrow random \in \{1, ..., n\}$;
  if $\sum_{l' \in \mathcal{L}_j} r_{jl'} < maxResAct$ **then**
    $l \longleftarrow random \in \mathcal{L}_j$;
    if $r_{jl} < maxResSkill$ **then**
      $r_{jl} \longleftarrow r_{jl} + 1$;
      if *skill requirements of the activity are met (i.e., a feasible flow is found)* **then**
        $\rho \longleftarrow \rho + 1$;
      else
        $r_{jl} \longleftarrow r_{jl} - 1$;
      end
    end
  end
end